The LuaTEX-ja package

The LuaTEX-ja project team

July 11, 2013

Contents

I User's manual **3**

1 Introduction **3**
1.1 Backgrounds . . . 3
1.2 Major Changes from pTEX . . . 3
1.3 Notations . . . 4
1.4 About the project . . . 4

2 Getting Started **5**
2.1 Installation . . . 5
2.2 Cautions . . . 6
2.3 Using in plain TEX . . . 6
2.4 Using in LATEX . . . 6

3 Changing Fonts **7**
3.1 plain TEX and LATEX 2_ε . . . 7
3.2 fontspec . . . 8
3.3 Preset . . . 8
3.4 `\CID`, `\UTF` and macros in `otf` package . . . 10

4 Changing Parameters **10**
4.1 Editing the range of **JAchar**s . . . 10
4.2 kanjiskip and xkanjiskip . . . 12
4.3 Insertion Setting of xkanjiskip . . . 12
4.4 Shifting Baseline . . . 13

II Reference **13**

5 Font Metric and Japanese Font **13**
5.1 `\jfont` . . . 13
5.2 Prefix `psft` . . . 15
5.3 Structure of JFM file . . . 15
5.4 Math Font Family . . . 17
5.5 Callbacks . . . 17

6 Parameters **18**
6.1 `\ltjsetparameter` . . . 18
6.2 List of Parameters . . . 19

7 Other Control Sequences **20**
7.1 Control Sequences for Compatibility . . . 20
7.2 `\inhibitglue` . . . 20

8 Control Sequences for LATEX 2_ε **21**
8.1 Patch for NFSS2 . . . 21

9 Extensions **22**
9.1 `luatexja-fontspec.sty` . . . 22
9.2 `luatexja-otf.sty` . . . 22
9.3 `luatexja-adjust.sty` . . . 22

III Implementations **22**

10 Storing Parameters **23**
10.1 Used Dimensions, Attributes and whatsit nodes . . . 23
10.2 Stack System of LuaTEX-ja . . . 24

11 Linebreak after Japanese Character **25**
11.1 Reference: Behavior in pTEX . . . 25
11.2 Behavior in LuaTEX-ja . . . 25

12 Patch for the `listings` package **26**

References **27**

A The category code of non-kanji characters defined in JIS X 0213 **28**

B Package versions used in this document **31**

This documentation is far from complete. It may have many grammatical (and contextual) errors. Also, several parts are written in Japanese only.

Part I
User's manual

1 Introduction

The LuaTEX-ja package is a macro package for typesetting high-quality Japanese documents when using LuaTEX.

1.1 Backgrounds

Traditionally, ASCII pTEX, an extension of TEX, and its derivatives are used to typeset Japanese documents in TEX. pTEX is an engine extension of TEX: so it can produce high-quality Japanese documents without using very complicated macros. But this point is a mixed blessing: pTEX is left behind from other extensions of TEX, especially ε-TEX and pdfTEX, and from changes about Japanese processing in computers (*e.g.*, the UTF-8 encoding).

Recently extensions of pTEX, namely upTEX (Unicode-implementation of pTEX) and ε-pTEX (merging of pTEX and ε-TEX extension), have developed to fill those gaps to some extent, but gaps still exist.

However, the appearance of LuaTEX changed the whole situation. With using Lua 'callbacks', users can customize the internal processing of LuaTEX. So there is no need to modify sources of engines to support Japanese typesetting: to do this, we only have to write Lua scripts for appropriate callbacks.

1.2 Major Changes from pTEX

The LuaTEX-ja package is under much influence of pTEX engine. The initial target of development was to implement features of pTEX. However, *LuaTEX-ja is not a just porting of pTEX; unnatural specifications/behaviors of pTEX were not adopted*.

The followings are major changes from pTEX:

- A Japanese font is a tuple of a 'real' font, a Japanese font metric (**JFM**, for short).
- In pTEX, a line break after Japanese character is ignored (and doesn't yield a space), since line breaks (in source files) are permitted almost everywhere in Japanese texts. However, LuaTEX-ja doesn't have this function completely, because of a specification of LuaTEX.
- The insertion process of glues/kerns between two Japanese characters and between a Japanese character and other characters (we refer glues/kerns of both kinds as **JAglue**) is rewritten from scratch.
  - As LuaTEX's internal character handling is 'node-based' (*e.g.*, `of{}fice` doesn't prevent ligatures), the insertion process of **JAglue** is now 'node-based'.
  - Furthermore, nodes between two characters which have no effects in line break (*e.g.*, `\special` node) and kerns from italic correction are ignored in the insertion process.
  - *Caution: due to above two points, many methods which did for the dividing the process of the insertion of **JAglue** in pTEX are not effective anymore.* In concrete terms, the following two methods are not effective anymore:

    ```
    \hskip2\zw ちょ{}っと\hskip2\zw ちょ\/っと
    ```

    If you want to do so, please put an empty hbox between it instead:

    ```
    \hskip2\zw ちょ\hbox{}っと
    ```

  - In the process, two Japanese fonts which only differ in their 'real' fonts are identified.
- At the present, vertical typesetting (*tategaki*), is not supported in LuaTEX-ja.

For detailed information, see Part III.

1.3 Notations

In this document, the following terms and notations are used:

- Characters are divided into two types:
  - **JAchar**: standing for characters which used in Japanese typesetting, such as Hiragana, Katakana, Kanji and other Japanese punctuation marks.
  - **ALchar**: standing for all other characters like alphabets.

  We say 'alphabetic fonts' for fonts used in **ALchar**, and 'Japanese fonts' for fonts used in **JAchar**.
- A word in a sans-serif font (like prebreakpenalty) means an internal parameter for Japanese typesetting, and it is used as a key in `\ltjsetparameter` command.
- A word in typewriter font with underline (like `fontspec`) means a package or a class of LaTeX.
- In this document, natural numbers start from 0.

1.4 About the project

■Project Wiki Project Wiki is under construction.

- http://sourceforge.jp/projects/luatex-ja/wiki/FrontPage%28en%29 (English)
- http://sourceforge.jp/projects/luatex-ja/wiki/FrontPage (Japanese)
- http://sourceforge.jp/projects/luatex-ja/wiki/FrontPage%28zh%29 (Chinese)

This project is hosted by SourceForge.JP.

■Members

- Hironori KITAGAWA
- Yusuke KUROKI
- Tomoaki HONDA
- Kazuki MAEDA
- Noriyuki ABE
- Shuzaburo SAITO
- Takayuki YATO
- Munehiro YAMAMOTO
- MA Qiyuan

2 Getting Started

2.1 Installation

To install the LuaTeX-ja package, you will need:

- LuaTeX beta-0.74.0 (or later)
- luaotfload v2.2
- luatexbase v0.6 (2013/05/04)
- xunicode v0.981 (2011/09/09)
- adobemapping (Adobe cmap and pdfmapping files)

From this version of LuaTeX-ja, TeX Live 2012 (or older version) is no longer supported, since LuaTeX binary and luaotfload is updated in TeX Live 2013. And conversely, older versions of LuaTeX-ja (20130318.1 or earlier) don't work in TeX Live 2013.

Now LuaTeX-ja is available from the following archive and distributions:

- CTAN (in the `macros/luatex/generic/luatexja` directory)
- MiKTeX (in `luatexja.tar.lzma`)
- TeX Live (in `texmf-dist/tex/luatex/luatexja`)
- W32TeX (in `luatexja.tar.xz`)

If you are using TeX Live 2013, you can install LuaTeX-ja from TeX Live manager (`tlmgr`):

```
$ tlmgr install luatexja
```

If you want to install manually, do the following instruvtions:

1. Download the source archive, by one of the following method. At the present, LuaTeX-ja has no *stable* release.

   - Copy the Git repository:

     ```
     $ git clone git://git.sourceforge.jp/gitroot/luatex-ja/luatexja.git
     ```

   - Download the `tar.gz` archive of HEAD in the `master` branch from

     http://git.sourceforge.jp/view?p=luatex-ja/luatexja.git;a=snapshot;h=HEAD;sf=tgz.

   Note that the `master` branch, and hence the archive in CTAN, are not updated frequently; the forefront of development is not the `master` branch.

2. Extract the archive. You will see `src/` and several other sub-directories. But only the contents in `src/` are needed to work LuaTeX-ja.

3. If you downloaded this package from CTAN, you have to run following commands to generate classes and `ltj-kinsoku.lua` (the file which stores default "*kinsoku*" parameters):

   ```
   $ cd src
   $ lualatex ltjclasses.ins
   $ lualatex ltjsclasses.ins
   $ lualatex ltjltxdoc.ins
   $ luatex   ltj-kinsoku_make.tex
   ```

   Note that `*.{dtx,ins}` and `ltj-kinsoku_make.tex` are not needed in regular use.

4. Copy all the contents of `src/` into one of your `TEXMF` tree. `TEXMF/tex/luatex/luatexja/` is an example location. If you cloned entire Git repository, making a symbolic link of `src/` instead copying is also good.

5. If `mktexlsr` is needed to update the file name database, make it so.

2.2 Cautions

- The encoding of your source file must be UTF-8. No other encodings, such as EUC-JP or Shift-JIS, are not supported.
- LuaTEX-ja is very slower than pTEX. Using LuaJITTEXslightly improve the situation.

2.3 Using in plain TEX

To use LuaTEX-ja in plain TEX, simply put the following at the beginning of the document:

```
\input luatexja.sty
```

This does minimal settings (like `ptex.tex`) for typesetting Japanese documents:

- The following 6 Japanese fonts are preloaded:

classification	font name	'10 pt'	'7 pt'	'5 pt'
mincho	Ryumin-Light	`\tenmin`	`\sevenmin`	`\fivemin`
gothic	GothicBBB-Medium	`\tengt`	`\sevengt`	`\fivegt`

  - It is widely accepted that fonts 'Ryumin-Light' and 'GothicBBB-Medium' aren't embedded into PDF files, and a PDF reader substitute them by some external Japanese fonts (*e.g.*, Ryumin-Light is substituted with Kozuka Mincho in Adobe Reader). We adopt this custom to the default setting.
  - A character in an alphabetic font is generally smaller than a Japanese font in the same size. So actual size specification of these Japanese fonts is in fact smaller than that of alphabetic fonts, namely scaled by 0.962216.

- The amount of glue that are inserted between a **JAchar** and an **ALchar** (the parameter xkanjiskip) is set to

$$(0.25 \cdot 0.962216 \cdot 10\,\mathrm{pt})^{+1\,\mathrm{pt}}_{-1\,\mathrm{pt}} = 2.40554\,\mathrm{pt}^{+1\,\mathrm{pt}}_{-1\,\mathrm{pt}}.$$

2.4 Using in LATEX

■LATEX 2$_\varepsilon$ Using in LATEX 2$_\varepsilon$ is basically same. To set up the minimal environment for Japanese, you only have to load `luatexja.sty`:

```
\usepackage{luatexja}
```

It also does minimal settings (counterparts in pLATEX are `plfonts.dtx` and `pldefs.ltx`):

- `JY3` is the font encoding for Japanese fonts (in horizontal direction).
  When vertical typesetting is supported by LuaTEX-ja in the future, `JT3` will be used for vertical fonts.
- Traditionally, Japanese documents use two typeface category: *mincho* (明朝体) and *gothic* (**ゴシック体**). *mincho* is used in the main text, while *gothic* is used in the headings or for emphasis.

classification			family name
mincho (明朝体)	`\textmc{...}`	`{\mcfamily ...}`	`\mcdefault`
gothic (**ゴシック体**)	`\textgt{...}`	`{\gtfamily ...}`	`\gtdefault`

- By default, the following fonts are used for *mincho* and *gothic*:

classification	family name	`\mdseries`	`\bfseries`	scale
mincho (明朝体)	`mc`	Ryumin-Light	GothicBBB-Medium	0.962216
gothic (**ゴシック体**)	`gt`	GothicBBB-Medium	GothicBBB-Medium	0.962216

Note that the bold series in both family are same as the medium series of *gothic* family. This is a convention in pLaTeX. This is trace that there were only 2 fonts (these are Ryumin-Light and GothicBBB-Medium) in early years of DTP. There is no italic nor slanted shape for these `mc` and `gt`.

- Japanese characters in math mode are typeset by the font family `mc`.

However, above settings are not sufficient for Japanese-based documents. To typeset Japanese-based documents, you are better to use class files other than `article.cls`, `book.cls`, and so on. At the present, we have the counterparts of `jclasses` (standard classes in pLaTeX) and `jsclasses` (classes by Haruhiko Okumura), namely, `ltjclasses` and `ltjsclasses`.

3 Changing Fonts

3.1 plain TeX and LaTeX 2ε

■plain TeX To change Japanese fonts in plain TeX, you must use the control sequence `\jfont`. So please see Subsection 5.1.

■LaTeX 2ε (NFSS2) For LaTeX 2ε, LuaTeX-ja adopted most of the font selection system of pLaTeX 2ε (in `plfonts.dtx`).

- Commands `\fontfamily`, `\fontseries`, `\fontshape` and `\selectfont` can be used to change attributes of Japanese fonts.

	encoding	**family**	**series**	**shape**	**selection**
alphabetic fonts	`\romanencoding`	`\romanfamily`	`\romanseries`	`\romanshape`	`\useroman`
Japanese fonts	`\kanjiencoding`	`\kanjifamily`	`\kanjiseries`	`\kanjishape`	`\usekanji`
both	—	–	`\fontseries`	`\fontshape`	—
auto select	`\fontencoding`	`\fontfamily`	—	—	`\usefont`

`\fontencoding{<encoding>}` changes the encoding of alphabetic fonts or Japanese fonts depending on the argument. For example, `\fontencoding{JY3}` changes the encoding of Japanese fonts to `JY3` and `\fontencoding{T1}` changes the encoding of alphabetic fonts to `T1`. `\fontfamily` also changes the family of Japanese fonts, alphabetic fonts, *or both*. For detail, see Subsection 8.1.

- For defining a Japanese font family, use `\DeclareKanjiFamily` instead of `\DeclareFontFamily`. However, in the present implementation, using `\DeclareFontFamily` doesn't cause any problem.

■Remark: Japanese Characters in Math Mode Since pTeX supports Japanese characters in math mode, there are sources like the following:

```
1 $f_{高温}$~($f_{\text{high temperature}}$).
2 \[ y=(x-1)^2+2\quad よって\quad y>0 \]
3 $5\in 素:=\{\,p\in\mathbb N:\text{$p$ is a prime
    }\,\}$.
```

$f_{高温}$ ($f_{\text{high temperature}}$).

$$y=(x-1)^2+2 \quad よって \quad y>0$$

$5 \in 素 := \{\, p \in \mathbb{N} : p \text{ is a prime} \,\}$.

We (the project members of LuaTeX-ja) think that using Japanese characters in math mode are allowed if and only if these are used as identifiers. In this point of view,

- The lines 1 and 2 above are not correct, since '高温' in above is used as a textual label, and 'よって' is used as a conjunction.
- However, the line 3 is correct, since '素' is used as an identifier.

Hence, in our opinion, the above input should be corrected as:

```
$f_{\text{高温}}$~%
($f_{\text{high temperature}}$).
\[ y=(x-1)^2+2\quad
  \mathrel{\text{よって}}\quad y>0 \]
$5\in 素:=\{\,p\in\mathbb N:\text{$p$ is a prime
    }\,\}$.
```

$f_{\text{高温}}$ ($f_{\text{high temperature}}$).

$$y=(x-1)^2+2 \quad \text{よって} \quad y>0$$

$5 \in 素 := \{\, p \in \mathbb{N} : p \text{ is a prime} \,\}$.

We also believe that using Japanese characters as identifiers is rare, hence we don't describe how to change Japanese fonts in math mode in this chapter. For the method, please see Subsection 5.4.

3.2 fontspec

To coexist with the fontspec package, it is needed to load luatexja-fontspec package in the preamble. This additional package automatically loads luatexja and fontspec package, if needed.

In luatexja-fontspec package, the following 7 commands are defined as counterparts of original commands in the fontspec package:

Japanese fonts alphabetic fonts	\jfontspec \fontspec	\setmainjfont \setmainfont	\setsansjfont \setsansfont	\newjfontfamily \newfontfamily
Japanese fonts alphabetic fonts	\newjfontface \newfontface	\defaultjfontfeatures \defaultfontfeatures	\addjfontfeatures \addfontfeatures	

```
\fontspec[Numbers=OldStyle]{LMSans10-Regular}
\jfontspec{IPAexMincho}
JIS~X~0213:2004→辻

\jfontspec[CJKShape=JIS1990]{IPAexMincho}
JIS~X~0208:1990→辻
```

JIS X 0213:2004 →辻
JIS X 0208:1990 →辻

Note that there is no command named \setmonojfont, since it is popular for Japanese fonts that nearly all Japanese glyphs have same widths. Also note that the kerning feature is set off by default in these 7 commands, since this feature and **JAglue** will clash (see 5.1).

3.3 Preset

To use standard Japanese font settings easily, one can load luatexja-preset package with several options. This package provides functions in a part of otf package and a part of PXchfon package by Takayuki Yato, and loads luatexja-fontspec internally.

■General options

`nodeluxe` Use one-weighted *mincho* and *gothic* font families. This means that \mcfamily\bfseries, \gtfamily\bfseries and \gtfamily\mdseries use the same font. *This option is enabled by default.*

`deluxe` Use *mincho* with two weights (medium and bold), *gothic* with three weights (medium, bold and heavy), and *rounded gothic*[1]. The heavy weight of *gothic* can be used by "changing the family" \gtebfamily, because fontspec package can handle only medium (\mdseries) and bold (\bfseries).

`expert` Use horizontal kana alternates, and define a control sequence \rubyfamily to use kana characters designed for ruby.

`bold` Use bold gothic as bold mincho.

`90jis` Use 90JIS glyph variants if possible.

`jis2004` Use JIS2004 glyph variants if possible.

`jis` Use the JFM jfm-jis.lua, instead of jfm-ujis.lua, which is the default JFM of LuaTeX-ja.

Note that 90jis and jis2004 only affect with *mincho*, *gothic* (and possibly *rounded gothic*) defined by this package. We didn't taken accound of when both 90jis and jis2004 are specified.

[1] Provided by \mgfamily, because *rounded gothic* is called *maru gothic* (丸ゴシック) in Japanese.

■Kozuka fonts There is not 'Kozuka Maru Gothic', therefore Kozuka Gothic H is used as a substitute for *rounded gothic*.

	`kozuka-pro`	`kozuka-pr6`	`kozuka-pr6n`
mincho medium	Kozuka Mincho Pro R	Kozuka Mincho ProVI R	Kozuka Mincho Pr6N R
mincho bold	Kozuka Mincho Pro B	Kozuka Mincho ProVI B	Kozuka Mincho Pr6N B
gothic medium			
without `deluxe`	Kozuka Gothic Pro M	Kozuka Gothic ProVI M	Kozuka Gothic Pr6N M
with `deluxe`	Kozuka Gothic Pro R	Kozuka Gothic ProVI R	Kozuka Gothic Pr6N R
gothic bold	Kozuka Gothic Pro B	Kozuka Gothic ProVI B	Kozuka Gothic Pr6N B
gothic heavy	Kozuka Gothic Pro H	Kozuka Gothic ProVI H	Kozuka Gothic Pr6N H
(rounded gothic)	Kozuka Gothic Pro H	Kozuka Gothic ProVI H	Kozuka Gothic Pr6N H

■Hiragino and Morisawa

	`hiragino-pro`	`hiragino-pron`
mincho medium	Hiragino Mincho Pro W3	Hiragino Mincho Pr6N W3
mincho bold	Hiragino Mincho Pro W6	Hiragino Mincho Pr6N W6
gothic medium		
without `deluxe`	Hiragino Kaku Gothic Pro W6	Hiragino Kaku Gothic ProN W6
with `deluxe`	Hiragino Kaku Gothic Pro W3	Hiragino Kaku Gothic ProN W3
gothic bold	Hiragino Kaku Gothic Pro W6	Hiragino Kaku Gothic ProN W6
gothic heavy	Hiragino Kaku Gothic Std W8	Hiragino Kaku Gothic StdN W8
rounded gothic	Hiragino Maru Gothic Pro W4	Hiragino Maru Gothic ProN W4

	`morisawa-pro`	`morisawa-pr6n`
mincho medium	Ryumin Pro L-KL	Ryumin Pr6N L-KL
mincho bold	Futo Min A101 Pro Bold	Futo Min A101 Pr6N Bold
gothic medium	Chu Gothic BBB Pro Med	Chu Gothic BBB Pr6N Med
gothic bold	Futo Go B101 Pro Bold	Futo Go B101 Pr6N Bold
gothic heavy	Midashi Go Pro MB31	Midashi Go Pr6N MB31
rounded gothic	Jun Pro 101	Jun Pr6N 101

■Settings for single weight Next, we describe settings for using only single weight. In four settings below, we use same fonts for medium and bold (and heavy) weights. (Hence `\mcfamily\bfseries` and `\mcfamily\mdseries` yields same Japanese fonts, even if `deluxe` option is also specified).

	`noembed`	`ipa`	`ipaex`	`ms`
mincho	Ryumin-Light (non-embedded)	IPAMincho	IPAexMincho	MS Mincho
gothic	GothicBBB-Medium (non-embedded)	IPAGothic	IPAexGothic	MS Gothic

■Using HG fonts We can use HG fonts bundled with Microsoft Office for realizing multiple weights in Japanese fonts.

	`ipa-hg`	`ipaex-hg`	`ms-hg`
mincho medium	IPAMincho	IPAexMincho	MS Mincho
mincho bold		HG Mincho E	
Gothic medium			
without `deluxe`	IPAGothic	IPAexGothic	MS Gothic
with `jis2004`	IPAGothic	IPAexGothic	MS Gothic
otherwise		HG Gothic M	
gothic bold		HG Gothic E	
gothic heavy		HG Soei Kaku Gothic UB	
rounded gothic		HG Maru Gothic PRO	

Note that HG Mincho E, HG Gothic E, HG Soei Kaku Gothic UB and HG Maru Gothic PRO are internally specified by:

default by font name (`HGMinchoE`, etc.).

`90jis` by filename (`hgrme.ttc`, `hgrge.ttc`, `hgrsgu.ttc`, `hgrsmp.ttf`).

`jis2004` by filename (`hgrme04.ttc`, `hgrge04.ttc`, `hgrsgu04.ttc`, `hgrsmp04.ttf`).

3.4 `\CID`, `\UTF` and macros in `otf` package

Under pLaTeX, `otf` package (developed by Shuzaburo Saito) is used for typesetting characters which is in Adobe-Japan1-6 CID but not in JIS X 0208. Since this package is widely used, LuaTeX-ja supports some of functions in `otf` package. If you want to use these functions, load `luatexja-otf` package.

```
\jfontspec{KozMinPr6N-Regular.otf}
森\UTF{9DD7}外と内田百\UTF{9592}とが\UTF{9AD9}島
    屋に行く。

\CID{7652}飾区の\CID{13706}野家，
\CID{1481}城市，葛西駅，
高崎と\CID{8705}\UTF{FA11}

\aj半角{はんかくカタカナ}
```

森鷗外と内田百閒とが髙島屋に行く。
葛飾区の吉野家，葛城市，葛西駅，高崎と髙﨑
はんかくｶﾀｶﾅ

4 Changing Parameters

There are many parameters in LuaTeX-ja. And due to the behavior of LuaTeX, most of them are not stored as internal register of TeX, but as an original storage system in LuaTeX-ja. Hence, to assign or acquire those parameters, you have to use commands `\ltjsetparameter` and `\ltjgetparameter`.

4.1 Editing the range of JAchars

To edit the range of **JAchar**s, you have to assign a non-zero natural number which is less than 217 to the character range first. This can be done by using `\ltjdefcharrange`. For example, the next line assigns whole characters in Supplementary Ideographic Plane and the character '漢' to the range number 100.

```
\ltjdefcharrange{100}{"20000-"2FFFF,`漢}
```

This assignment of numbers to ranges are always global, so you should not do this in the middle of a document.

If some character has been belonged to some non-zero numbered range, this will be overwritten by the new setting. For example, whole SIP belong to the range 4 in the default setting of LuaTeX-ja, and if you specify the above line, then SIP will belong to the range 100 and be removed from the range 4.

After assigning numbers to ranges, the jacharrange parameter can be used to customize which character range will be treated as ranges of **JAchar**s, as the following line (this is just the default setting of LuaTeX-ja):

```
\ltjsetparameter{jacharrange={-1, +2, +3, -4, -5, +6, +7, +8}}
```

The argument to jacharrange parameter is a list of integer. Negative integer $-n$ in the list means that 'the characters that belong to range n are treated as **ALchar**', and positive integer $+n$ means that 'the characters that belong to range n are treated as **JAchar**'.

■**Default Setting** LuaTeX-ja predefines eight character ranges for convenience. They are determined from the following data:

- Blocks in Unicode 6.0.
- The `Adobe-Japan1-UCS2` mapping between a CID Adobe-Japan1-6 and Unicode.

Table 1. Unicode blocks in predefined character range 3.

U+2000–U+206F	General Punctuation	U+2070–U+209F	Superscripts and Subscripts
U+20A0–U+20CF	Currency Symbols	U+20D0–U+20FF	Comb. Diacritical Marks for Symbols
U+2100–U+214F	Letterlike Symbols	U+2150–U+218F	Number Forms
U+2190–U+21FF	Arrows	U+2200–U+22FF	Mathematical Operators
U+2300–U+23FF	Miscellaneous Technical	U+2400–U+243F	Control Pictures
U+2500–U+257F	Box Drawing	U+2580–U+259F	Block Elements
U+25A0–U+25FF	Geometric Shapes	U+2600–U+26FF	Miscellaneous Symbols
U+2700–U+27BF	Dingbats	U+2900–U+297F	Supplemental Arrows-B
U+2980–U+29FF	Misc. Mathematical Symbols-B	U+2B00–U+2BFF	Miscellaneous Symbols and Arrows

- The PXbase bundle for upTEX by Takayuki Yato.

Now we describe these eight ranges. The alphabet 'J' or 'A' after the number shows whether characters in the range is treated as **JAchar**s or not by default. These settings are similar to the `prefercjk` settings defined in `PXbase` bundle.

Range 8^{J} Symbols in the intersection of the upper half of ISO 8859-1 (Latin-1 Supplement) and JIS X 0208 (a basic character set for Japanese). This character range consists of the following characters:

- § (U+00A7, Section Sign)
- ¨ (U+00A8, Diaeresis)
- ° (U+00B0, Degree sign)
- ± (U+00B1, Plus-minus sign)
- ´ (U+00B4, Spacing acute)
- ¶ (U+00B6, Paragraph sign)
- × (U+00D7, Multiplication sign)
- ÷ (U+00F7, Division Sign)

Range 1^{A} Latin characters that some of them are included in Adobe-Japan1-6. This range consist of the following Unicode ranges, *except characters in the range 8 above*:

- U+0080–U+00FF: Latin-1 Supplement
- U+0100–U+017F: Latin Extended-A
- U+0180–U+024F: Latin Extended-B
- U+0250–U+02AF: IPA Extensions
- U+02B0–U+02FF: Spacing Modifier Letters
- U+0300–U+036F: Combining Diacritical Marks
- U+1E00–U+1EFF: Latin Extended Additional

Range 2^{J} Greek and Cyrillic letters. JIS X 0208 (hence most of Japanese fonts) has some of these characters.

- U+0370–U+03FF: Greek and Coptic
- U+0400–U+04FF: Cyrillic
- U+1F00–U+1FFF: Greek Extended

Range 3^{J} Punctuations and Miscellaneous symbols. The block list is indicated in Table 1.

Range 4^{A} Characters usually not in Japanese fonts. This range consists of almost all Unicode blocks which are not in other predefined ranges. Hence, instead of showing the block list, we put the definition of this range itself:

```
\ltjdefcharrange{4}{%
   "500-"10FF, "1200-"1DFF, "2440-"245F, "27C0-"28FF, "2A00-"2AFF,
  "2C00-"2E7F, "4DC0-"4DFF, "A4D0-"A82F, "A840-"ABFF, "FB00-"FE0F,
  "FE20-"FE2F, "FE70-"FEFF, "10000-"1FFFF, "E000-"F8FF} % non-Japanese
```

Range 5^{A} Surrogates and Supplementary Private Use Areas.

Range 6^{J} Characters used in Japanese. The block list is indicated in Table 2.

Range 7^{J} Characters used in CJK languages, but not included in Adobe-Japan1-6. The block list is indicated in Table 3.

Table 2. Unicode blocks in predefined character range 6.

U+2460–U+24FF	Enclosed Alphanumerics	U+2E80–U+2EFF	CJK Radicals Supplement
U+3000–U+303F	CJK Symbols and Punctuation	U+3040–U+309F	Hiragana
U+30A0–U+30FF	Katakana	U+3190–U+319F	Kanbun
U+31F0–U+31FF	Katakana Phonetic Extensions	U+3200–U+32FF	Enclosed CJK Letters and Months
U+3300–U+33FF	CJK Compatibility	U+3400–U+4DBF	CJK Unified Ideographs Extension A
U+4E00–U+9FFF	CJK Unified Ideographs	U+F900–U+FAFF	CJK Compatibility Ideographs
U+FE10–U+FE1F	Vertical Forms	U+FE30–U+FE4F	CJK Compatibility Forms
U+FE50–U+FE6F	Small Form Variants	U+20000–U+2FFFF	(Supplementary Ideographic Plane)

Table 3. Unicode blocks in predefined character range 7.

U+1100–U+11FF	Hangul Jamo	U+2F00–U+2FDF	Kangxi Radicals
U+2FF0–U+2FFF	Ideographic Description Characters	U+3100–U+312F	Bopomofo
U+3130–U+318F	Hangul Compatibility Jamo	U+31A0–U+31BF	Bopomofo Extended
U+31C0–U+31EF	CJK Strokes	U+A000–U+A48F	Yi Syllables
U+A490–U+A4CF	Yi Radicals	U+A830–U+A83F	Common Indic Number Forms
U+AC00–U+D7AF	Hangul Syllables	U+D7B0–U+D7FF	Hangul Jamo Extended-B

4.2 kanjiskip and xkanjiskip

JAglue is divided into the following three categories:

- Glues/kerns specified in JFM. If `\inhibitglue` is issued around a Japanese character, this glue will not be inserted at the place.
- The default glue which inserted between two **JAchar**s (kanjiskip).
- The default glue which inserted between a **JAchar** and an **ALchar** (xkanjiskip).

The value (a skip) of kanjiskip or xkanjiskip can be changed as the following.

```
\ltjsetparameter{kanjiskip={0pt plus 0.4pt minus 0.4pt},
                 xkanjiskip={0.25\zw plus 1pt minus 1pt}}
```

It may occur that JFM contains the data of 'ideal width of kanjiskip' and/or 'ideal width of xkanjiskip'. To use these data from JFM, set the value of kanjiskip or xkanjiskip to `\maxdimen`.

4.3 Insertion Setting of xkanjiskip

It is not desirable that xkanjiskip is inserted into every boundary between **JAchar**s and **ALchar**s. For example, xkanjiskip should not be inserted after opening parenthesis (*e.g.*, compare '(あ' and '(あ'). LuaTEX-ja can control whether xkanjiskip can be inserted before/after a character, by changing jaxspmode for **JAchar**s and alxspmode parameters **ALchar**s respectively.

```
1 \ltjsetparameter{jaxspmode={`あ,preonly},
    alxspmode={`\!,postonly}}
2 pあq い!う
```

p あq い! う

The second argument `preonly` means 'the insertion of xkanjiskip is allowed before this character, but not after'. the other possible values are `postonly`, `allow` and `inhibit`.

jaxspmode and alxspmode use a same table to store the parameters on the current version. Therefore, line 1 in the code above can be rewritten as follows:

```
\ltjsetparameter{alxspmode={`あ,preonly}, jaxspmode={`\!,postonly}}
```

One can use also numbers to specify these two parameters (see Subsection 6.2).

If you want to enable/disable all insertions of kanjiskip and xkanjiskip, set autospacing and autoxspacing parameters to `true`/`false`, respectively.

4.4 Shifting Baseline

To make a match between a Japanese font and an alphabetic font, sometimes shifting of the baseline of one of the pair is needed. In pTEX, this is achieved by setting `\ybaselineshift` to a non-zero length (the baseline of alphabetic fonts is shifted below). However, for documents whose main language is not Japanese, it is good to shift the baseline of Japanese fonts, but not that of alphabetic fonts. Because of this, LuaTEX-ja can independently set the shifting amount of the baseline of alphabetic fonts (yalbaselineshift parameter) and that of Japanese fonts (yjabaselineshift parameter).

```
\vrule width 150pt height 0.4pt depth 0pt\hskip
    -120pt
\ltjsetparameter{yjabaselineshift=0pt,
    yalbaselineshift=0pt}abcあいう
\ltjsetparameter{yjabaselineshift=5pt,
    yalbaselineshift=2pt}abcあいう
```

abc あいう abc あいう

Here the horizontal line in above is the baseline of a line.

There is an interesting side-effect: characters in different size can be vertically aligned center in a line, by setting two parameters appropriately. The following is an example (beware the value is not well tuned):

```
xyz漢字
{\scriptsize
  \ltjsetparameter{yjabaselineshift=-1pt,
    yalbaselineshift=-1pt}
  XYZひらがな
}abcかな
```

xyz 漢字 XYZ ひらがな abc かな

Part II
Reference

5 Font Metric and Japanese Font

5.1 `\jfont`

To load a font as a Japanese font, you must use the `\jfont` instead of `\font`, while `\jfont` admits the same syntax used in `\font`. LuaTEX-ja automatically loads `luaotfload` package, so TrueType/OpenType fonts with features can be used for Japanese fonts:

```
\jfont\tradgt={file:KozMinPr6N-Regular.otf:script=latn;%
  +trad;-kern;jfm=ujis} at 14pt
\tradgt 当／体／医／区
```

當／體／醫／區

Note that the defined control sequence (`\tradgt` in the example above) using `\jfont` is not a *font_def* token, hence the input like `\fontname\tradgt` causes a error. We denote control sequences which are defined in `\jfont` by ⟨*jfont_cs*⟩.

■JFM As noted in Introduction, a JFM has measurements of characters and glues/kerns that are automatically inserted for Japanese typesetting. The structure of JFM will be described in the next subsection. At the calling of `\jfont`, you must specify which JFM will be used for this font by the following keys:

`jfm=`⟨*name*⟩ Specify the name of JFM. If specified JFM has not been loaded, LuaTEX-ja search and load a file named `jfm-`⟨*name*⟩`.lua`.

The following JFMs are shipped with LuaTEX-ja:

`jfm-ujis.lua` A standard JFM in LuaTEX-ja. This JFM is based on `upnmlminr-h.tfm`, a metric for UTF/OTF package that is used in upTEX. When you use the `luatexja-otf` package, you should use this JFM.

Table 4. Differences between JFMs shipped with LuaTEX-ja

	jfm-ujis.lua	jfm-jis.lua	jfm-min.lua
Example 1[4]	◆◆◆◆◆◆◆◆ ある日モモちゃんがお使いで迷子になって泣きました.	◆◆◆◆◆◆◆◆ ある日モモちゃんがお使いで迷子になって泣きました.	◆◆◆◆◆◆◆◆ ある日モモちゃんがお使いで迷子になって泣きました.
Example 2	ちょっと！何	ちょっと！何	ちょっと！何
Bounding Box	漢	漢	漢

`jfm-jis.lua` A counterpart for `jis.tfm`, 'JIS font metric' which is widely used in pTEX. A major difference of `jfm-ujis.lua` and this `jfm-jis.lua` is that most characters under `jfm-ujis.lua` are square-shaped, while that under `jfm-jis.lua` are horizontal rectangles.

`jfm-min.lua` A counterpart for `min10.tfm`, which is one of the default Japanese font metric shipped with pTEX. There are notable difference between this JFM and other 2 JFMs, as shown in Table 4.

`jfmvar=`⟨*string*⟩ Sometimes there is a need that ….

```
\ltjsetparameter{differentjfm=both}
\jfont\F=file:KozMinPr6N-Regular.otf:jfm=ujis
\jfont\G=file:KozGoPr6N-Medium.otf:jfm=ujis
\jfont\H=file:KozGoPr6N-Medium.otf:jfm=ujis;jfmvar=hoge

\F ) {\G 【】} ( % halfwidth space
   ) {\H 『』} ( % fullwidth space

\ltjsetparameter{differentjfm=paverage}
```

) 【】 ( ) 『』 (

■Note: kern feature Some fonts have information for inter-glyph spacing. However, this information is not well-compatible with LuaTEX-ja. More concretely, this kerning space from this information are inserted *before* the insertion process of **JAglue**, and this causes incorrect spacing between two characters when both a glue/kern from the data in the font and it from JFM are present.

- You should specify `-kern` in `jfont` when you want to use other font features, such as `script=...`.
- If you want to use Japanese fonts in proportional width, and use information from this font, use `jfm-prop.lua` for its JFM, and …. TODO: kanjiskip?

■`extend` and `slant` The following setting can be specified as OpenType font features:

`extend=`⟨*extend*⟩ expand the font horizontally by ⟨*extend*⟩.

`slant=`⟨*slant*⟩ slant the font.

Note that LuaTEX-ja doesn't adjust JFMs by these `extend` and `slant` settings; you have to write new JFMs on purpose. For example, the following example uses the standard JFM `jfm-ujis.lua`, hence letter-spacing and the width of italic correction are not correct:

```
\jfont\E=file:KozMinPr6N-Regular.otf:extend=1.5;jfm=ujis
\E あいうえお

\jfont\S=file:KozMinPr6N-Regular.otf:slant=1;jfm=ujis
\S あいう\/ABC
```

あいうえお
*あいう*ABC

5.2 Prefix `psft`

Besides '`file:`' and '`name:`' prefixes, one can use '`psft:`' prefix in `\jfont` (and `\font`), to specify a 'name-only' Japanese font which will not be embedded to PDF. Typical use of this prefix is to specify the 'standard' Japanese fonts, namely, 'Ryumin-Light' and 'GothicBBB-Medium'.

OpenType font features, such as '`+jp90`', have no meaning in 'name-only' fonts using this '`psft:`' prefix. This is because we can't expect what fonts are actually used by the PDF reader. Note that `extend` and `slant` settings (see above) are supported with `psft` prefix, because they are only simple linear transformations.

■`cid` key The default font defined by using `psft:` prefix is for Japanese typesetting; it is Adobe-Japan1-6 CID-keyed font. One can specify `cid` key to use other CID-keyed non-embedded fonts for Chinese or Korean typesetting.

```
\jfont\testJ={psft:Ryumin-Light:cid=Adobe-Japan1-6;jfm=jis}     % Japanese
\jfont\testD={psft:Ryumin-Light:jfm=jis}                       % default value is Adobe-Japan1-6
\jfont\testC={psft:AdobeMingStd-Light:cid=Adobe-CNS1-6;jfm=jis} % Traditional Chinese
\jfont\testG={psft:SimSun:cid=Adobe-GB1-5;jfm=jis}             % Simplified Chinese
\jfont\testK={psft:Batang:cid=Adobe-Korea1-2;jfm=jis}          % Korean
```

Note that the code above specifies `jfm-jis.lua`, which is for Japanese fonts, as JFM for Chinese and Korean fonts.

At present, LuaTEX-ja supports only 4 values written in the sample code above. Specifying other values, e.g.,

```
\jfont\test={psft:Ryumin-Light:cid=Adobe-Japan2;jfm=jis}
```

produces the following error:

```
! Package luatexja Error: bad cid key `Adobe-Japan2'.

See the luatexja package documentation for explanation.
Type  H <return>  for immediate help.
<to be read again>
                   \par
l.78

? h
I couldn't find any non-embedded font information for the CID
`Adobe-Japan2'. For now, I'll use `Adobe-Japan1-6'.
Please contact the LuaTeX-ja project team.
?
```

5.3 Structure of JFM file

A JFM file is a Lua script which has only one function call:

```
luatexja.jfont.define_jfm { ... }
```

Real data are stored in the table which indicated above by `{ ... }`. So, the rest of this subsection are devoted to describe the structure of this table. Note that all lengths in a JFM file are floating-point numbers in design-size unit.

`dir=`⟨*direction*⟩ (required)

The direction of JFM. At the present, only `'yoko'` is supported.

`zw=`⟨*length*⟩ (required)

The amount of the length of the 'full-width'.

`zh=`⟨*length*⟩ (required)

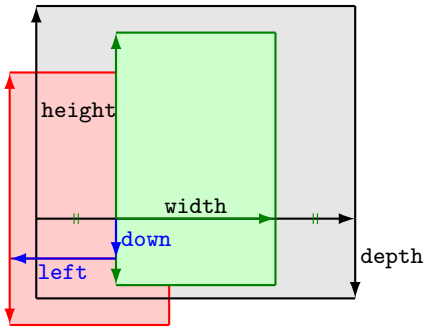
The amount of the 'full-height' (height + depth).

`kanjiskip={`⟨*natural*⟩`,` ⟨*stretch*⟩`,` ⟨*shrink*⟩`}` (optional)

This field specifies the 'ideal' amount of kanjiskip. As noted in Subsection 4.2, if the parameter kanjiskip is `\maxdimen`, the value specified in this field is actually used (if this field is not specified in JFM, it is regarded as 0 pt). Note that ⟨*stretch*⟩ and ⟨*shrink*⟩ fields are in design-size unit too.

`xkanjiskip={`⟨*natural*⟩`,` ⟨*stretch*⟩`,` ⟨*shrink*⟩`}` (optional)

Like the `kanjiskip` field, this field specifies the 'ideal' amount of xkanjiskip.



Consider a node containing Japanese character whose value of the `align` field is `'middle'`.

- The black rectangle is a frame of the node. Its width, height and depth are specified by JFM.
- Since the `align` field is `'middle'`, the 'real' glyph is centered horizontally (the green rectangle).
- Furthermore, the glyph is shifted according to values of fields `left` and `down`. The ultimate position of the real glyph is indicated by the red rectangle.

Figure 1. The position of the 'real' glyph.

■Character classes Besides from above fields, a JFM file have several sub-tables those indices are natural numbers. The table indexed by $i \in \omega$ stores information of 'character class' i. At least, the character class 0 is always present, so each JFM file must have a sub-table whose index is `[0]`. Each sub-table (its numerical index is denoted by i) has the following fields:

`chars={`⟨*character*⟩`, ...}` (required except character class 0)

This field is a list of characters which are in this character type i. This field is optional if $i = 0$, since all **JAchar** which do not belong any character classes other than 0 are in the character class 0 (hence, the character class 0 contains most of **JAchar**s). In the list, character(s) can be specified in the following form:

- a Unicode code point
- the character itself (as a Lua string, like `'あ'`)
- a string like `'あ*'` (the character followed by an asterisk)
- several "imaginary" characters (We will describe these later.)

`width=`⟨*length*⟩`, height=`⟨*length*⟩`, depth=`⟨*length*⟩`, italic=`⟨*length*⟩ (required)

Specify width of characters in character class i, height, depth and the amount of italic correction. All characters in character class i are regarded that its width, height and depth are as values of these fields. But there is one exception: if `'prop'` is specified in `width` field, width of a character becomes that of its 'real' glyph

`left=`⟨*length*⟩`, down=`⟨*length*⟩`, align=`⟨*align*⟩

These fields are for adjusting the position of the 'real' glyph. Legal values of `align` field are `'left'`, `'middle'` and `'right'`. If one of these 3 fields are omitted, `left` and `down` are treated as 0, and `align` field is treated as `'left'`. The effects of these 3 fields are indicated in Figure 1.

In most cases, `left` and `down` fields are 0, while it is not uncommon that the `align` field is `'middle'` or `'right'`. For example, setting the `align` field to `'right'` is practically needed when the current character class is the class for opening delimiters'.

`kern={[`j`]=`⟨*kern*⟩`, [`j'`]={`⟨*kern*⟩`, [`⟨*ratio*⟩`]}, ...}`

`glue={[`j`]={`⟨*width*⟩`,` ⟨*stretch*⟩`,` ⟨*shrink*⟩`, [`⟨*priority*⟩`], [`⟨*ratio*⟩`]}, ...}`

`end_stretch=`⟨*kern*⟩

`end_shrink=`⟨*kern*⟩

■Imaginary characters As described before, you can specify several 'imaginary characters' in `chars` field. The most of these characters are regarded as the characters of class 0 in pTEX. As a result, LuaTEX-ja can control typesetting finer than pTEX. The following is the list of 'imaginary characters':

`'boxbdd'` The beginning/ending of a horizontal box, and the beginning of a noindented paragraph.

`'parbdd'` The beginning of an (indented) paragraph.

Table 5. Control sequences for Japanese math fonts

Japanese fonts	**alphabetic fonts**
`\jfam` $\in [0, 256)$	`\fam`
jatextfont ={⟨*jfam*⟩,⟨*jfont_cs*⟩}	`\textfont`⟨*fam*⟩=⟨*font_cs*⟩
jascriptfont ={⟨*jfam*⟩,⟨*jfont_cs*⟩}	`\scriptfont`⟨*fam*⟩=⟨*font_cs*⟩
jascriptscriptfont ={⟨*jfam*⟩,⟨*jfont_cs*⟩}	`\scriptscriptfont`⟨*fam*⟩=⟨*font_cs*⟩

`'jcharbdd'` A boundary between **JAchar** and anything else (such as **ALchar**, kern, glue, …).

−1 The left/right boundary of an inline math formula.

■Porting JFM from pTEX …

5.4 Math Font Family

TEX handles fonts in math formulas by 16 font families[2], and each family has three fonts: `\textfont`, `\scriptfont` and `\scriptscriptfont`.

LuaTEX-ja's handling of Japanese fonts in math formulas is similar; Table 5 shows counterparts to TEX's primitives for math font families. There is no relation between the value of `\fam` and that of `\jfam`; with appropriate settings, you can set both `\fam` and `\jfam` to the same value.

5.5 Callbacks

Like LuaTEX itself, LuaTEX-ja also has callbacks. These callbacks can be accessed via `luatexbase.add_to_callback` function and so on, as other callbacks.

`luatexja.load_jfm` callback With this callback you can overwrite JFMs. This callback is called when a new JFM is loaded.

```
function (<table> jfm_info, <string> jfm_name)
  return <table> new_jfm_info
end
```

The argument `jfm_info` contains a table similar to the table in a JFM file, except this argument has `chars` field which contains character codes whose character class is not 0.

An example of this callback is the `ltjarticle` class, with forcefully assigning character class 0 to `'parbdd'` in the JFM `jfm-min.lua`.

`luatexja.define_font` callback This callback and the next callback form a pair, and you can assign letters which don't have fixed code points in Unicode to non-zero character classes. This `luatexja.define_font` callback is called just when new Japanese font is loaded.

```
function (<table> jfont_info, <number> font_number)
  return <table> new_jfont_info
end
```

You may assume that `jfont_info` has the following fields:

`size_cache` A table which contains the information of a JFM, and *this table must not be changed*. The contents of this table are similar to that which is written is the JFM file, but the following differ:

- There is a `chars` table, …
- The value in `zw`, `zh`, `kanjiskip`, `xkanjiskip` fields are now scaled by real font size, and in scaled-pont unit.
- ...

[2]Omega, Aleph, LuaTEX and ε-(u)pTEX can handles 256 families, but an external package is needed to support this in plain TEX and LATEX.

- There is no `dir` field in this table.

`var` The value specified in `jfmvar=...` at a call of `\jfont`.

The returned table `new_jfont_info` also should include these two fields. The `font_number` is a font number.

A good example of this and the next callbacks is the `luatexja-otf` package, supporting `"AJ1-xxx"` form for Adobe-Japan1 CID characters in a JFM. This callback doesn't replace any code of LuaTeX-ja.

`luatexja.find_char_class` callback This callback is called just when LuaTeX-ja is trying to determine which character class a character `chr_code` belongs. A function used in this callback should be in the following form:

```
function (<number> char_class, <table> jfont_info, <number> chr_code)
  if char_class~=0 then return char_class
  else
    ....
    return (<number> new_char_class or 0)
  end
end
```

The argument `char_class` is the result of LuaTeX-ja's default routine or previous function calls in this callback, hence this argument may not be 0. Moreover, the returned `new_char_class` should be as same as `char_class` when `char_class` is not 0, otherwise you will overwrite the LuaTeX-ja's default routine.

`luatexja.set_width` callback This callback is called when LuaTeX-ja is trying to encapsule a **JAchar** *glyph_node*, to adjust its dimension and position.

```
function (<table> shift_info, <table> jfont_info, <number> char_class)
  return <table> new_shift_info
end
```

The argument `shift_info` and the returned `new_shift_info` have `down` and `left` fields, which are the amount of shifting down/left the character in a scaled-point.

A good example is `test/valign.lua`. After loading this file, the vertical position of glyphs is automatically adjusted; the ratio (height : depth) of glyphs is adjusted to be that of letters in the character class 0. For example, suppose that

- The setting of the JFM: (height) $= 88x$, (depth) $= 12x$ (the standard values of Japanese OpenType fonts);
- The value of the real font: (height) $= 28y$, (depth) $= 5y$ (the standard values of Japanese TrueType fonts).

Then, the position of glyphs is shifted up by

$$\frac{88x}{88x+12x}(28y+5y)-28y=\frac{26}{25}y=1.04y.$$

6 Parameters

6.1 `\ltjsetparameter`

As noted before, `\ltjsetparameter` and `\ltjgetparameter` are control sequences for accessing most parameters of LuaTeX-ja. One of the main reason that LuaTeX-ja didn't adopted the syntax similar to that of pTeX (*e.g.*, `\prebreakpenalty`` =10000`) is the position of `hpack_filter` callback in the source of LuaTeX, see Section 10.

`\ltjsetparameter` and `\ltjglobalsetparameter` are control sequences for assigning parameters. These take one argument which is a ⟨*key*⟩=⟨*value*⟩ list. Allowed keys are described in the next subsection. The difference between `\ltjsetparameter` and `\ltjglobalsetparameter` is only the scope of assignment; `\ltjsetparameter` does a local assignment and `\ltjglobalsetparameter` does a global one. They also obey the value of `\globaldefs`, like other assignment.

`\ltjgetparameter` is for acquiring parameters. It always takes a parameter name as first argument, and also takes the additional argument—a character code, for example—in some cases.

```
\ltjgetparameter{differentjfm},
\ltjgetparameter{autospacing},                    paverage, 1, 10000.
\ltjgetparameter{prebreakpenalty}{`) }.
```

The return value of `\ltjgetparameter` *is always a string*. This is outputted by `tex.write()`, so any character other than space ‘ ’ (U+0020) has the category code 12 (other), while the space has 10 (space).

6.2 List of Parameters

The following is the list of parameters which can be specified by the `\ltjsetparameter` command. [`\cs`] indicates the counterpart in pTEX, and symbols beside each parameter has the following meaning:

- No mark: values at the end of the paragraph or the hbox are adopted in the whole paragraph/hbox.
- ‘∗’ : local parameters, which can change everywhere inside a paragraph/hbox.
- ‘†’: assignments are always global.

jcharwidowpenalty =⟨*penalty*⟩ [`\jcharwidowpenalty`] Penalty value for suppressing orphans. This penalty is inserted just after the last **JAchar** which is not regarded as a (Japanese) punctuation mark.

kcatcode =`{`⟨*chr_code*⟩`,`⟨*natural number*⟩`}` An additional attributes which each character whose character code is ⟨*chr_code*⟩ has. At the present version, the lowermost bit of ⟨*natural number*⟩ indicates whether the character is considered as a punctuation mark (see the description of jcharwidowpenalty above).

prebreakpenalty =`{`⟨*chr_code*⟩`,`⟨*penalty*⟩`}` [`\prebreakpenalty`]

postbreakpenalty =`{`⟨*chr_code*⟩`,`⟨*penalty*⟩`}` [`\postbreakpenalty`]

jatextfont =`{`⟨*jfam*⟩`,`⟨*jfont_cs*⟩`}` [`\textfont` in TEX]

jascriptfont =`{`⟨*jfam*⟩`,`⟨*jfont_cs*⟩`}` [`\scriptfont` in TEX]

jascriptscriptfont =`{`⟨*jfam*⟩`,`⟨*jfont_cs*⟩`}` [`\scriptscriptfont` in TEX]

yjabaselineshift =⟨*dimen*⟩∗

yalbaselineshift =⟨*dimen*⟩∗ [`\ybaselineshift`]

jaxspmode =`{`⟨*chr_code*⟩`,`⟨*mode*⟩`}` Setting whether inserting xkanjiskip is allowed before/after a **JAchar** whose character code is ⟨*chr_code*⟩. The followings are allowed for ⟨*mode*⟩:

0, `inhibit` Insertion of xkanjiskip is inhibited before the character, nor after the character.

1, `preonly` Insertion of xkanjiskip is allowed before the character, but not after.

2, `postonly` Insertion of xkanjiskip is allowed after the character, but not before.

3, `allow` Insertion of xkanjiskip is allowed both before the character and after the character. This is the default value.

This parameter is similar to the `\inhibitxspcode` primitive of pTEX, but not compatible with `\inhibitxspcode`.

alxspmode =`{`⟨*chr_code*⟩`,`⟨*mode*⟩`}` [`\xspcode`]

Setting whether inserting xkanjiskip is allowed before/after a **ALchar** whose character code is ⟨*chr_code*⟩. The followings are allowed for ⟨*mode*⟩:

0, `inhibit` Insertion of xkanjiskip is inhibited before the character, nor after the character.

1, `preonly` Insertion of xkanjiskip is allowed before the character, but not after.

2, `postonly` Insertion of xkanjiskip is allowed after the character, but not before.

3, `allow` Insertion of xkanjiskip is allowed before the character and after the character. This is the default value.

Note that parameters jaxspmode and alxspmode share a common table, hence these two parameters are synonyms of each other.

autospacing =⟨*bool*⟩* [\autospacing]

autoxspacing =⟨*bool*⟩* [\autoxspacing]

kanjiskip =⟨*skip*⟩ [\kanjiskip]

xkanjiskip =⟨*skip*⟩ [\xkanjiskip]

differentjfm =⟨*mode*⟩† Specify how glues/kerns between two **JAchar**s whose JFM (or size) are different. The allowed arguments are the followings:

average

both

large

small

pleft

pright

paverage

jacharrange =⟨*ranges*⟩*

kansujichar ={⟨*digit*⟩, ⟨*chr_code*⟩} [\kansujichar]

7 Other Control Sequences

7.1 Control Sequences for Compatibility

The following control sequences are implemented for compatibility with pTEX. Note that these don't support JIS X 0213, but only JIS X 0208.

\kuten

\jis

\euc

\sjis

\ucs

\kansuji

7.2 \inhibitglue

\inhibitglue suppresses the insertion of **JAglue**. The following is an example, using a special JFM that there will be a glue between the beginning of a box and 'あ', and also between 'あ' and 'ウ'.

```
\jfont\g=file:KozMinPr6N-Regular.otf:jfm=test \g
\fbox{\hbox{あウあ\inhibitglue ウ}}
\inhibitglue\par\noindent あ1
\par\inhibitglue\noindent あ2
\par\noindent\inhibitglue あ3
\par\hrule\noindent あoff\inhibitglue ice
```

```
あ　ウあウ
あ　1
あ　2
あ　3
あ　office
```

With the help of this example, we remark the specification of \inhibitglue:

- The call of \inhibitglue in the (internal) vertical mode is simply ignored.
- The call of \inhibitglue in the (restricted) horizontal mode is only effective on the spot; does not get over boundary of paragraphs. Moreover, \inhibitglue cancels ligatures and kernings, as shown in the last line of above example.
- The call of \inhibitglue in math mode is just ignored.

8 Control Sequences for LaTeX 2ε

8.1 Patch for NFSS2

As described in Subsection 2.4, LuaTeX-ja simply adopted `plfonts.dtx` in pLaTeX 2ε for the Japanese patch for NFSS2. For an convenience, we will describe control sequences which are not described in Subsection 3.1.

`\DeclareYokoKanjiEncoding{`⟨*encoding*⟩`}{`⟨*text-settings*⟩`}{`⟨*math-settings*⟩`}`

In NFSS2 under LuaTeX-ja, distinction between alphabetic font families and Japanese font families are only made by their encodings. For example, encodings OT1 and T1 are for alphabetic font families, and a Japanese font family cannot have these encodings. This command defines a new encoding scheme for Japanese font family (in horizontal direction).

`\DeclareKanjiEncodingDefaults{`⟨*text-settings*⟩`}{`⟨*math-settings*⟩`}`

`\DeclareKanjiSubstitution{`⟨*encoding*⟩`}{`⟨*family*⟩`}{`⟨*series*⟩`}{`⟨*shape*⟩`}`

`\DeclareErrorKanjiFont{`⟨*encoding*⟩`}{`⟨*family*⟩`}{`⟨*series*⟩`}{`⟨*shape*⟩`}{`⟨*size*⟩`}`

The above 3 commands are just the counterparts for `DeclareFontEncodingDefaults` and others.

`\reDeclareMathAlphabet{`⟨*unified-cmd*⟩`}{`⟨*al-cmd*⟩`}{`⟨*ja-cmd*⟩`}`

`\DeclareRelationFont{`⟨*ja-encoding*⟩`}{`⟨*ja-family*⟩`}{`⟨*ja-series*⟩`}{`⟨*ja-shape*⟩`}`
`{`⟨*al-encoding*⟩`}{`⟨*al-family*⟩`}{`⟨*al-series*⟩`}{`⟨*al-shape*⟩`}`

This command sets the 'accompanied' alphabetic font family (given by the latter 4 arguments) with respect to a Japanese font family given by the former 4 arguments.

`\SetRelationFont`

This command is almost same as `\DeclareRelationFont`, except that this command does a local assignment, where `\DeclareRelationFont` does a global assignment.

`\userelfont`

Change current alphabetic font encoding/family/… to the 'accompanied' alphabetic font family with respect to current Japanese font family, which was set by `\DeclareRelationFont` or `\SetRelationFont`. Like `\fontfamily`, `\selectfont` is required to take an effect.

`\adjustbaseline`

…

`\fontfamily{`⟨*family*⟩`}`

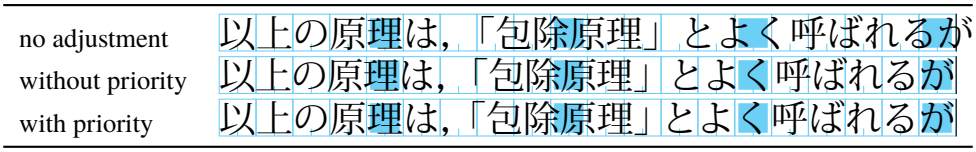
As in LaTeX 2ε, this command changes current font family (alphabetic, Japanese, *or both*) to ⟨*family*⟩. Which family will be changed is determined as follows:

- Let current encoding scheme for Japanese fonts be ⟨*ja-enc*⟩. Current Japanese font family will be changed to ⟨*family*⟩, if one of the following two conditions is met:
  - The family ⟨*family*⟩ under the encoding ⟨*ja-enc*⟩ has been already defined by `\DeclareKanijFamily`.
  - A font definition named ⟨*ja-enc*⟩⟨*family*⟩`.fd` (the file name is all lowercase) exists.
- Let current encoding scheme for alphabetic fonts be ⟨*al-enc*⟩. For alphabetic font family, the criterion as above is used.
- There is a case which none of the above applies, that is, the font family named ⟨*family*⟩ doesn't seem to be defined neither under the encoding ⟨*ja-enc*⟩, nor under ⟨*al-enc*⟩. In this case, the default family for font substitution is used for alphabetic and Japanese fonts. Note that current encoding will not be set to ⟨*family*⟩, unlike the original implementation in LaTeX.

As closing this subsection, we shall introduce an example of `\SetRelationFont` and `\userelfont`:

```
\makeatletter
\SetRelationFont{JY3}{\k@family}{m}{n}{OT1}{pag}{m}{n}
  % \k@family: current Japanese font family
\userelfont\selectfont あいうabc
```

あいう abc



Note: the value of kanjiskip is 0 pt$^{+1/5\text{ em}}_{-1/5\text{ em}}$ in this figure, for making the difference obvious.

Figure 2. Line adjustment

9 Extensions

9.1 `luatexja-fontspec.sty`

As described in Subsection 3.2, this optional package provides the counterparts for several commands defined in the `fontspec` package. In addition to 'font features' in the original `fontspec`, the following 'font features' specifications are allowed for the commands of Japanese version:

`CID=`⟨*name*⟩

`JFM=`⟨*name*⟩

`JFM-var=`⟨*name*⟩

These 3 font features correspond to `cid`, `jfm` and `jfmvar` keys for `\jfont` respectively. `CID` is effective only when with `NoEmbed` described below. See Subsections 5.1 and 5.2 for details.

`NoEmbed` By specifying this font feature, one can use 'name-only' Japanese font which will not be embedded in the output PDF file. See Subsection 5.2.

9.2 `luatexja-otf.sty`

This optional package supports typesetting characters in Adobe-Japan1. `luatexja-otf.sty` offers the following 2 low-level commands:

`\CID{`⟨*number*⟩`}` Typeset a character whose CID number is ⟨*number*⟩.

`\UTF{`⟨*hex_number*⟩`}` Typeset a character whose character code is ⟨*hex_number*⟩ (in hexadecimal). This command is similar to `\char"`⟨*hex_number*⟩, but please remind remarks below.

■Remarks Characters by `\CID` and `\UTF` commands are different from ordinary characters in the following points:

- Always treated as **JAchar**s.
- Processing codes for supporting OpenType features (*e.g.*, glyph replacement and kerning) by the `luaotfload` package is not performed to these characters.

■Additional Syntax of JFM `luatexja-otf.sty` extends the syntax of JFM; the entries of `chars` table in JFM now allows a string in the form `'AJ1-xxx'`, which stands for the character whose CID number in Adobe-Japan1 is `xxx`.

9.3 `luatexja-adjust.sty`

...

Part III
Implementations

10 Storing Parameters

10.1 Used Dimensions, Attributes and whatsit nodes

Here the following is the list of dimensions and attributes which are used in LuaTEX-ja.

`\jQ` (dimension) `\jQ` is equal to 1 Q = 0.25 mm, where 'Q' (also called '級') is a unit used in Japanese phototypesetting. So one should not change the value of this dimension.

`\jH` (dimension) There is also a unit called '歯' which equals to 0.25 mm and used in Japanese phototypesetting. This `\jH` is a synonym of `\jQ`.

`\ltj@zw` (dimension) A temporal register for the 'full-width' of current Japanese font.

`\ltj@zh` (dimension) A temporal register for the 'full-height' (usually the sum of height of imaginary body and its depth) of current Japanese font.

`\jfam` (attribute) Current number of Japanese font family for math formulas.

`\ltj@curjfnt` (attribute) The font index of current Japanese font.

`\ltj@charclass` (attribute) The character class of Japanese *glyph_node*.

`\ltj@yablshift` (attribute) The amount of shifting the baseline of alphabetic fonts in scaled point (2^{-16} pt).

`\ltj@ykblshift` (attribute) The amount of shifting the baseline of Japanese fonts in scaled point (2^{-16} pt).

`\ltj@autospc` (attribute) Whether the auto insertion of kanjiskip is allowed at the node.

`\ltj@autoxspc` (attribute) Whether the auto insertion of xkanjiskip is allowed at the node.

`\ltj@icflag` (attribute) An attribute for distinguishing 'kinds' of a node. One of the following value is assigned to this attribute:

***italic* (1)** Glues from an italic correction (`\/`). This distinction of origins of glues (from explicit `\kern`, or from `\/`) is needed in the insertion process of xkanjiskip.

***packed* (2)**

***kinsoku* (3)** Penalties inserted for the word-wrapping process of Japanese characters (*kinsoku*).

***from_jfm* (6)** Glues/kerns from JFM.

***kanji_skip* (9)** Glues for kanjiskip.

***xkanji_skip* (10)** Glues for xkanjiskip.

***processed* (11)** Nodes which is already processed by ….

***ic_processed* (12)** Glues from an italic correction, but also already processed.

***boxbdd* (15)** Glues/kerns that inserted just the beginning or the ending of an hbox or a paragraph.

`\ltj@kcat`*i* (attribute) Where *i* is a natural number which is less than 7. These 7 attributes store bit vectors indicating which character block is regarded as a block of **JAchar**s.

Furthermore, LuaTEX-ja uses several 'user-defined' whatsit nodes for inrernal processing. All those nodes store a natural number (hence the node's `type` is 100).

`inhibitglue` Nodes for indicating that `\inhibitglue` is specified. The `value` field of these nodes doesn't matter.

`stack_marker` Nodes for LuaTEX-ja's stack system (see the next subsection). The `value` field of these nodes is current group.

`char_by_cid` Nodes for Japanese Characters which the callback process of `luaotfload` won't be applied, and the character code is stored in the `value` field. Each node having this `user_id` is converted to a 'glyph_node' *after* the callback process of luaotfload. This `user_id` is only used by the `luatexja-otf` package.

`begin_par` Nodes for indicating beginning of a paragraph. A paragraph which is started by `\item` in list-like environments has a horizontal box for its label before the actual contents. So …

These whatsits will be removed during the process of inserting **JAglue**s.

10.2 Stack System of LuaTEX-ja

■Background LuaTEX-ja has its own stack system, and most parameters of LuaTEX-ja are stored in it. To clarify the reason, imagine the parameter kanjiskip is stored by a skip, and consider the following source:

```
1 \ltjsetparameter{kanjiskip=0pt}ふがふが.%
2 \setbox0=\hbox{\ltjsetparameter{kanjiskip=5pt}ほ
    げほげ}
3 \box0.ぴよぴよ\par
```

ふがふが. ほ げ ほ げ. ぴよぴよ

As described in Subsection 6.2, the only effective value of kanjiskip in an hbox is the latest value, so the value of kanjiskip which applied in the entire hbox should be 5 pt. However, by the implementation method of LuaTEX, this '5 pt' cannot be known from any callbacks. In the `tex/packaging.w` (which is a file in the source of LuaTEX), there are the following codes:

```
void package(int c)
{
    scaled h;                    /* height of box */
    halfword p;                  /* first node in a box */
    scaled d;                    /* max depth */
    int grp;
    grp = cur_group;
    d = box_max_depth;
    unsave();
    save_ptr -= 4;
    if (cur_list.mode_field == -hmode) {
        cur_box = filtered_hpack(cur_list.head_field,
                                 cur_list.tail_field, saved_value(1),
                                 saved_level(1), grp, saved_level(2));
        subtype(cur_box) = HLIST_SUBTYPE_HBOX;
```

Notice that `unsave` is executed *before* `filtered_hpack` (this is where `hpack_filter` callback is executed): so '5 pt' in the above source is orphaned at `unsave`, and hence it can't be accessed from `hpack_filter` callback.

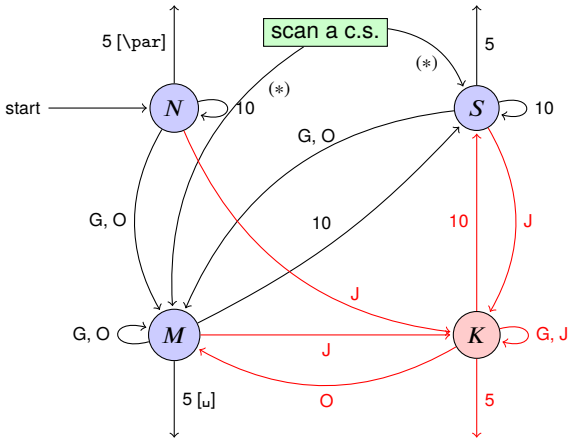
■The method The code of stack system is based on that in a post of Dev-luatex mailing list[3].

These are two TEX count registers for maintaining information: `\ltj@@stack` for the stack level, and `\ltj@@group@level` for the TEX's group level when the last assignment was done. Parameters are stored in one big table named `charprop_stack_table`, where `charprop_stack_table[`*i*`]` stores data of stack level *i*. If a new stack level is created by `\ltjsetparameter`, all data of the previous level is copied.

To resolve the problem mentioned in 'Background' above, LuaTEX-ja uses another thing: When a new stack level is about to be created, a whatsit node whose type, subtype and value are 44 (*user_defined*), 30112, and current group level respectively is appended to the current list (we refer this node by *stack_flag*). This enables us to know whether assignment is done just inside a hbox. Suppose that the stack level is s and the TEX's group level is t just after the hbox group, then:

- If there is no *stack_flag* node in the list of the hbox, then no assignment was occurred inside the hbox. Hence values of parameters at the end of the hbox are stored in the stack level s.
- If there is a *stack_flag* node whose value is $t+1$, then an assignment was occurred just inside the hbox group. Hence values of parameters at the end of the hbox are stored in the stack level $s+1$.

[3] `[Dev-luatex] tex.currentgrouplevel`, a post at 2008/8/19 by Jonathan Sauer.



G Beginning of group (usually `{`) and ending of group (usually `}`).

J Japanese characters.

5 *end-of-line* (usually `^^J`).

10 space (usually ␣).

O other characters, whose category code is in $\{3, 4, 6, 7, 8, 11, 12, 13\}$.

[␣], [\par] emits a space, or `\par`.

- We omitted about category codes 9 (*ignored*), 14 (*comment*) and 15 (*invalid*) from the above diagram. We also ignored the input like '`^^A`' or '`^^df`'.
- When a character whose category code is 0 (*escape character*) is seen by TEX, the input processor scans a control sequence (scan a c.s.). These paths are not shown in the above diagram.

  After that, the state is changed to State S (skipping blanks) in most cases, but to State M (middle of line) sometimes.

Figure 3. State transitions of pTEX's input processor.

- If there are *stack_flag* nodes but all of their values are more than $t + 1$, then an assignment was occurred in the box, but it is done is 'more internal' group. Hence values of parameters at the end of the hbox are stored in the stack level s.

Note that to work this trick correctly, assignments to `\ltj@@stack` and `\ltj@@group@level` have to be local always, regardless the value of `\globaldefs`. This problem is resolved by using `\directlua{tex.globaldefs=0}` (this assignment is local).

11 Linebreak after Japanese Character

11.1 Reference: Behavior in pTEX

In pTEX, a line break after a Japanese character doesn't emit a space, since words are not separated by spaces in Japanese writings. However, this feature isn't fully implemented in LuaTEX-ja due to the specification of callbacks in LuaTEX. To clarify the difference between pTEX and LuaTEX, We briefly describe the handling of a line break in pTEX, in this subsection.

pTEX's input processor can be described in terms of a finite state automaton, as that of TEX in Section 2.5 of [1]. The internal states are as follows:

- State N: new line
- State S: skipping spaces
- State M: middle of line
- State K: after a Japanese character

The first three states—N, S and M—are as same as TEX's input processor. State K is similar to state M, and is entered after Japanese characters. The diagram of state transitions are indicated in Figure 3. Note that pTEX doesn't leave state K after 'beginning/ending of a group' characters.

11.2 Behavior in LuaTEX-ja

States in the input processor of LuaTEX is the same as that of TEX, and they can't be customized by any callbacks. Hence, we can only use `process_input_buffer` and `token_filter` callbacks for to suppress a space by a line break which is after Japanese characters.

However, `token_filter` callback cannot be used either, since a character in category code 5 (end-of-line) is converted into an space token *in the input processor*. So we can use only the `process_input_buffer` callback. This means that suppressing a space must be done *just before* an input line is read.

Considering these situations, handling of an end-of-line in LuaTEX-ja are as follows:

> A character U+FFFFF (its category code is set to 14 (comment) by LuaTEX-ja) is appended to an input line, *before LuaTEX actually process it*, if and only if the following three conditions are satisfied:
>
> 1. The category code of `\endlinechar`[4] is 5 (end-of-line).
> 2. The category code of U+FFFFF itself is 14 (comment).
> 3. The input line matches the following 'regular expression':
>
> $$(\text{any char})^*(\textbf{JAchar})\big(\{\text{catcode} = 1\} \cup \{\text{catcode} = 2\}\big)^*$$

■**Remark** The following example shows the major difference from the behavior of pTEX:

```
\ltjsetparameter{autoxspacing=false}
\ltjsetparameter{jacharrange={-6}}xあ
y\ltjsetparameter{jacharrange={+6}}zあ
u
```

xyzあ u

- There is no space between 'x' and 'y', since the line 2 ends with a **JAchar** 'あ' (this 'あ' considered as an **JAchar** at the ending of line 1).
- There is no space between 'あ' (in the line 3) and 'u', since the line 3 ends with an **ALchar** (the letter 'あ' considered as an **ALchar** at the ending of line 2).

12 Patch for the `listings` package

It is well-known that the `listings` package outputs weird results for Japanese input. The `listings` package makes most of letters active and assigns output command for each letter [2]. But Japanese characters are not included in these activated letters. For pTEX series, there is no method to make Japanese characters active; a patch `jlisting.sty` [3] resolves the problem forcibly.

In LuaTEX-ja, the problem is resolved by using `process_input_buffer` callback. The callback function inserts the output command before each letter above U+0080. This method can omits the process to make all Japanese characters active (most of the activated characters are not used in many cases).

If `listings.sty` and LuaTEX-ja were loaded, then the patch `lltjp-listings.sty` is loaded automatically at `\begin{document}`.

■**Class of characters** Roughly speaking, the `listings` package processes input as follows:

1. Collects *letters* and *digits*, which can be used for the name of identifiers.
2. When reading an *other*, outputs the collected character string (with modification, if needed).
3. Collects *others*.
4. When reading a *letter* or a *digit*, outputs the collected character string.
5. Turns back to 1.

By the above process, line breaks inside of an identifier are blocked. A flag `\lst@ifletter` indicates whether the previous character can be used for the name of identifiers or not.

For Japanese characters, line breaks are permitted on both sides except for parentheses, dashes, etc. To process Japanese characters, The pacth `lltjp-listings.sty` introduces a new flag `\lst@ifkanji`, which indicates whether the previous character is Japanese character or not. For illustration, we introduce the following classes of character:

[4] Usually, it is ⟨return⟩ (whose character code is 13).

	Letter	Other	Kanji	Open	Close
`\lst@ifletter`	T	F	T	F	T
`\lst@ifkanji`	F	F	T	T	F
Meaning	identifier char	other alphabet	most of Japanese char	open paren	close paren

Note that *digits* in the `listings` package can be Letter or Other according to circumstances.

For example, let us consider the case an Open comes after a Letter. Since an Open represents Japanese open parenthesis, it is preferred to be permitted to insert line break after the Letter. Therefore, the collected character string is output in this case.

The following table summarizes $5 \times 5 = 25$ cases:

		Next				
		Letter	Other	Kanji	Open	Close
Prev	Letter	collects	outputs			collects
	Other	outputs	collects	outputs		collects
	Kanji	outputs				collects
	Open	collects				
	Close	outputs				collects

In the above table,

- "outputs" means to output the collected character string (i.e., line breaking is permitted there).
- "collects" means to append the next character to the collected character string (i.e., line breaking is prohibited there).

■Classification of characters Characters are classified according to jacharrange parameter (see Section 4.1):

- **ALchars** above U+0080 are Letter.
- **JAchars** are classified in the order as follows:
  1. Characters whose prebreakpenalty is greater than or equal to 0 are Open.
  2. Characters whose postbreakpenalty is greater than or equal to 0 are Close.
  3. Characters that don't satisfy the above two conditions are Kanji.

The width of halfwidth kana (U+FF61–U+FF9F) is same as the width of **ALchar**; the width of the other **JAchars** is double the width of **ALchar**.

The classification process is executed every time a character appears in listing environments.

References

[1] Victor Eijkhout, *TEX by Topic, A TEXnician's Reference*, Addison-Wesley, 1992.

[2] C. Heinz, B. Moses. The Listings Package.

[3] Thor Watanabe. Listings - MyTeXpert. http://mytexpert.sourceforge.jp/index.php?Listings

[4] 乙部厳己, min10 フォントについて. http://argent.shinshu-u.ac.jp/~otobe/tex/files/min10.pdf

[5] W3C Japanese Layout Task Force (ed), Requirements for Japanese Text Layout (W3C Working Group Note), 2011, 2012. http://www.w3.org/TR/jlreq/

[6] 日本工業規格 (Japanese Industrial Standard) JIS X 4051, 日本語文書の組版方法 (Formatting rules for Japanese documents), 1993, 1995, 2004.

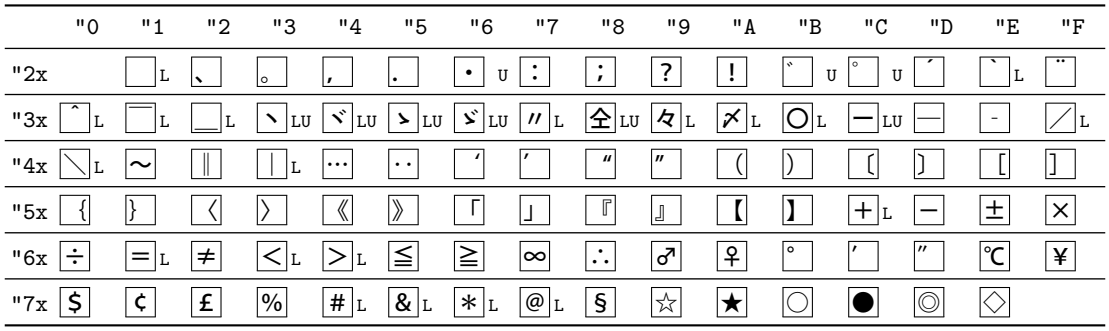
A The category code of non-kanji characters defined in JIS X 0213

In these tables, the default catcode (LuaTEX-ja) and kcatcode ((u)pTEX) of non-kanji characters defined in JIS X 0213 from row 1 to row 13 is summarized. Each character is printed as follows:

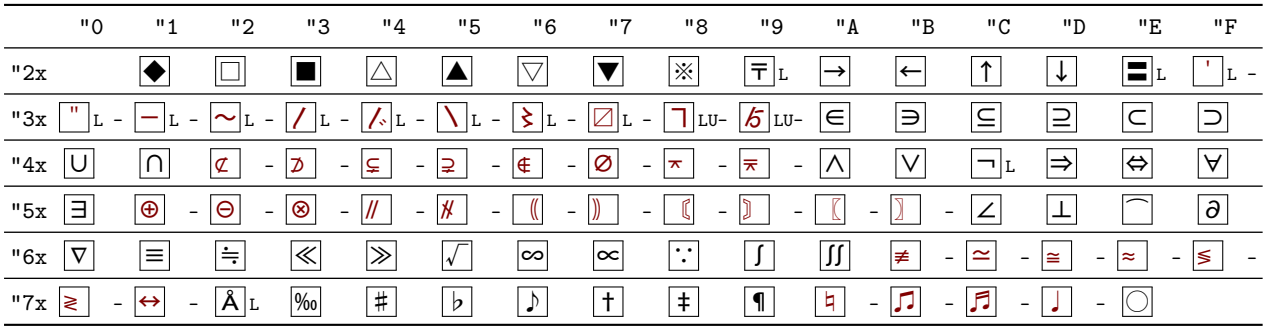
The tables are generated by using `\jis` command for characters included in JIS X 0208. Each character in the tables means:

- The background of a character regarded as **ALchar** in LuaTEX-ja is colored light blue.
- The first letter `L` means that the character is available for the name of a control sequence in XETEX and LuaTEX-ja (its catcode is 11).
- The second letter `U` means that the character is available for the name of a control sequence in upTEX (its kcatcode is 16 or 17). upTEX regards these characters as Japanese character.
- The third letter `P` means that the character is available for the name of a control sequence in pTEX (its kcatcode is 16 or 17).
- If the third letter is `-` (or the character is printed in red), the character is not included in JIS X 0208. Therefore, you can consider the character is not available in pTEX.
- The kana for Japanese syllable beginning with a voiced velar nasal consonant.kana in rows 4 and 5 are omitted.

Row 1



Row 2



Row 3

	"0	"1	"2	"3	"4	"5	"6	"7	"8	"9	"A	"B	"C	"D	"E	"F
"2x		▷ -	▶ -	◁ -	◀ -	↗ -	↘ -	↖ -	↙ -	⇄ -	⇨ -	⇦ -	⇧ -	⇩ -	⤴ -	⤵ -
"3x	0 LUP	1 LUP	2 LUP	3 LUP	4 LUP	5 LUP	6 LUP	7 LUP	8 LUP	9 LUP	⦿ -	◉ -	〽 L -	[illegible] L -	[illegible] L -	◦ -
"4x	• -	A LUP	B LUP	C LUP	D LUP	E LUP	F LUP	G LUP	H LUP	I LUP	J LUP	K LUP	L LUP	M LUP	N LUP	O LUP
"5x	P LUP	Q LUP	R LUP	S LUP	T LUP	U LUP	V LUP	W LUP	X LUP	Y LUP	Z LUP	∓ -	ℵ L -	ℏ L -	⅊ L -	ℓ L -
"6x	℧ -	a LUP	b LUP	c LUP	d LUP	e LUP	f LUP	g LUP	h LUP	i LUP	j LUP	k LUP	l LUP	m LUP	n LUP	o LUP
"7x	p LUP	q LUP	r LUP	s LUP	t LUP	u LUP	v LUP	w LUP	x LUP	y LUP	z LUP	= U-	− -	⧺ -	⧻ -	

Row 4

	"0	"1	"2	"3	"4	"5	"6	"7	"8	"9	"A	"B	"C	"D	"E	"F
"2x		ぁ LUP	あ LUP	ぃ LUP	い LUP	ぅ LUP	う LUP	ぇ LUP	え LUP	ぉ LUP	お LUP	か LUP	が LUP	き LUP	ぎ LUP	く LUP
"3x	ぐ LUP	け LUP	げ LUP	こ LUP	ご LUP	さ LUP	ざ LUP	し LUP	じ LUP	す LUP	ず LUP	せ LUP	ぜ LUP	そ LUP	ぞ LUP	た LUP
"4x	だ LUP	ち LUP	ぢ LUP	っ LUP	つ LUP	づ LUP	て LUP	で LUP	と LUP	ど LUP	な LUP	に LUP	ぬ LUP	ね LUP	の LUP	は LUP
"5x	ば LUP	ぱ LUP	ひ LUP	び LUP	ぴ LUP	ふ LUP	ぶ LUP	ぷ LUP	へ LUP	べ LUP	ぺ LUP	ほ LUP	ぼ LUP	ぽ LUP	ま LUP	み LUP
"6x	む LUP	め LUP	も LUP	ゃ LUP	や LUP	ゅ LUP	ゆ LUP	ょ LUP	よ LUP	ら LUP	り LUP	る LUP	れ LUP	ろ LUP	ゎ LUP	わ LUP
"7x	ゐ LUP	ゑ LUP	を LUP	ん LUP	ゔ LU-	ゕ LU-	ゖ LU-									

Row 5

	"0	"1	"2	"3	"4	"5	"6	"7	"8	"9	"A	"B	"C	"D	"E	"F
"2x		ァ LUP	ア LUP	ィ LUP	イ LUP	ゥ LUP	ウ LUP	ェ LUP	エ LUP	ォ LUP	オ LUP	カ LUP	ガ LUP	キ LUP	ギ LUP	ク LUP
"3x	グ LUP	ケ LUP	ゲ LUP	コ LUP	ゴ LUP	サ LUP	ザ LUP	シ LUP	ジ LUP	ス LUP	ズ LUP	セ LUP	ゼ LUP	ソ LUP	ゾ LUP	タ LUP
"4x	ダ LUP	チ LUP	ヂ LUP	ッ LUP	ツ LUP	ヅ LUP	テ LUP	デ LUP	ト LUP	ド LUP	ナ LUP	ニ LUP	ヌ LUP	ネ LUP	ノ LUP	ハ LUP
"5x	バ LUP	パ LUP	ヒ LUP	ビ LUP	ピ LUP	フ LUP	ブ LUP	プ LUP	ヘ LUP	ベ LUP	ペ LUP	ホ LUP	ボ LUP	ポ LUP	マ LUP	ミ LUP
"6x	ム LUP	メ LUP	モ LUP	ャ LUP	ヤ LUP	ュ LUP	ユ LUP	ョ LUP	ヨ LUP	ラ LUP	リ LUP	ル LUP	レ LUP	ロ LUP	ヮ LUP	ワ LUP
"7x	ヰ LUP	ヱ LUP	ヲ LUP	ン LUP	ヴ LUP	ヵ LUP	ヶ LUP									

Row 6

	"0	"1	"2	"3	"4	"5	"6	"7	"8	"9	"A	"B	"C	"D	"E	"F
"2x		Α L P	Β L P	Γ L P	Δ L P	Ε L P	Ζ L P	Η L P	Θ L P	Ι L P	Κ L P	Λ L P	Μ L P	Ν L P	Ξ L P	Ο L P
"3x	Π L P	Ρ L P	Σ L P	Τ L P	Υ L P	Φ L P	Χ L P	Ψ L P	Ω L P	♤ -	♠ -	♢ -	♦ -	♡ -	♥ -	♧ -
"4x	♣ -	α L P	β L P	γ L P	δ L P	ε L P	ζ L P	η L P	θ L P	ι L P	κ L P	λ L P	μ L P	ν L P	ξ L P	ο L P
"5x	π L P	ρ L P	σ L P	τ L P	υ L P	φ L P	χ L P	ψ L P	ω L P	ς L -	① -	② -	③ -	④ -	⑤ -	⑥ -
"6x	⑦ -	⑧ -	⑨ -	⑩ -	☖ -	☗ -	〠 L -	☎ -	☀ -	☁ -	☂ -	☃ -	♨ -	▱ -	ㇰ LU-	ㇱ LU-
"7x	ㇲ LU-	ㇳ LU-	ㇴ LU-	ㇵ LU-	ㇶ LU-	ㇷ LU-	ㇸ LU-	ㇹ LU-	ㇺ LU-	ㇻ LU-	ㇼ LU-	ㇽ LU-	ㇾ LU-	ㇿ LU-	ㇿ LU-	

Row 7

	"0	"1	"2	"3	"4	"5	"6	"7	"8	"9	"A	"B	"C	"D	"E	"F
"2x		А L	Б L	В L	Г L	Д L	Е L	Ё L	Ж L	З L	И L	Й L	К L	Л L	М L	Н L
"3x	О L	П L	Р L	С L	Т L	У L	Ф L	Х L	Ц L	Ч L	Ш L	Щ L	Ъ L	Ы L	Ь L	Э L
"4x	Ю L	Я L	-	-	-	-	-	-	-	-	-	-	-	-	-	-
"5x	-	а L	б L	в L	г L	д L	е L	ё L	ж L	з L	и L	й L	к L	л L	м L	н L
"6x	о L	п L	р L	с L	т L	у L	ф L	х L	ц L	ч L	ш L	щ L	ъ L	ы L	ь L	э L
"7x	ю L	я L	LU-	LU-	LU-	LU-	-	-	-	-	-	-	-	-	-	-

Row 8

	"0	"1	"2	"3	"4	"5	"6	"7	"8	"9	"A	"B	"C	"D	"E	"F
"2x																
"3x																
"4x		㉑ L	- ㉒ L	- ㉓ L	- ㉔ L	- ㉕ L	- ㉖ L	- ㉗ L	- ㉘ L	- ㉙ L	- ㉚ L	- ㉛ L	- ㉜ L	- ㉝ L	- ㉞ L	- ㉟ L	-
"5x	㊱ L	- ㊲ L	- ㊳ L	- ㊴ L	- ㊵ L	- ㊶ L	- ㊷ L	- ㊸ L	- ㊹ L	- ㊺ L	- ㊻ L	- ㊼ L	- ㊽ L	- ㊾ L	- ㊿ L	-	
"6x									-	-	-	-	!! -	?? -	?! -	!? -	Ă L -
"7x	ă L	- ĭ L	- Ḿ L	- ḿ L	- Ǹ L	- ǹ L	- Ǒ L	- ǒ L	- ǔ L	- ǔ L	- ǔ L	- ǔ L	- ǔ L	-			

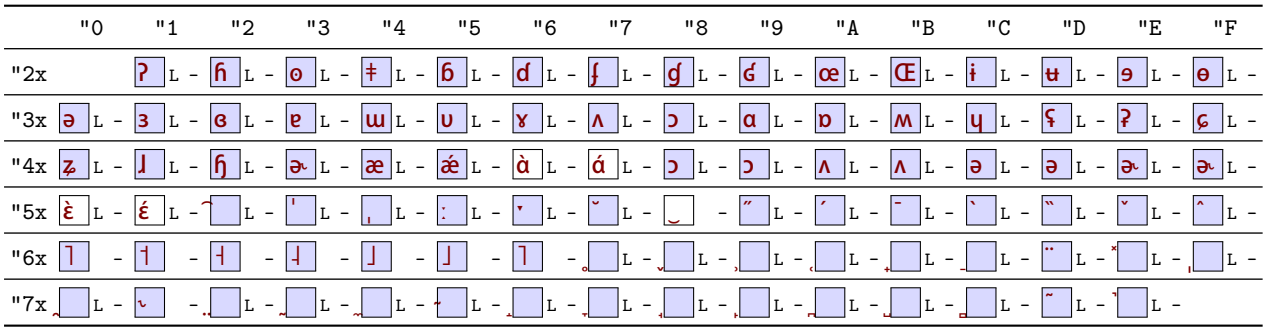
Row 9

	"0	"1	"2	"3	"4	"5	"6	"7	"8	"9	"A	"B	"C	"D	"E	"F
"2x		€ -	-	¡ -	¤ -	¦ -	© -	ª L -	« -	– -	® -	¯ -	² -	³ -	· -	¸ -
"3x	¹ -	º L -	» -	¼ -	½ -	¾ -	¿ -	À L -	Á L -	Â L -	Ã L -	Ä L -	Å L -	Æ L -	Ç L -	È L -
"4x	É L -	Ê L -	Ë L -	Ì L -	Í L -	Î L -	Ï L -	Ð L -	Ñ L -	Ò L -	Ó L -	Ô L -	Õ L -	Ö L -	Ø L -	Ù L -
"5x	Ú L -	Û L -	Ü L -	Ý L -	Þ L -	ß L -	à L -	á L -	â L -	ã L -	ä L -	å L -	æ L -	ç L -	è L -	é L -
"6x	ê L -	ë L -	ì L -	í L -	î L -	ï L -	ð L -	ñ L -	ò L -	ó L -	ô L -	õ L -	ö L -	ø L -	ù L -	ú L -
"7x	û L -	ü L -	ý L -	þ L -	ÿ L -	Ā L -	Ī L -	Ū L -	Ē L -	Ō L -	ā L -	ī L -	ū L -	ē L -	ō L -	

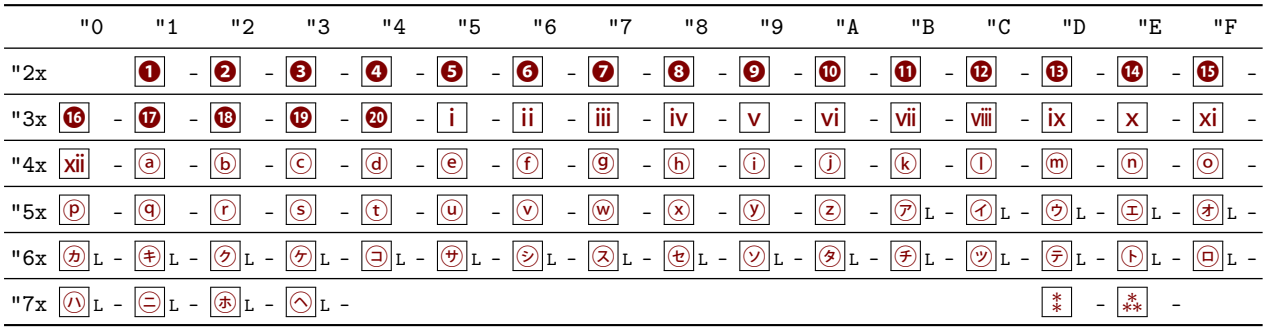
Row 10

	"0	"1	"2	"3	"4	"5	"6	"7	"8	"9	"A	"B	"C	"D	"E	"F
"2x		Ą L -	˘ -	Ł L -	Ľ L -	Ś L -	Š L -	Ş L -	Ť L -	Ź L -	Ž L -	Ż L -	ą L -	˛ -	ł L -	ľ L -
"3x	ś L -	ˇ L -	š L -	ş L -	ť L -	ź L -	˝ -	ž L -	ż L -	Ŕ L -	Ă L -	Ĺ L -	Ć L -	Č L -	Ę L -	Ě L -
"4x	Ď L -	Ń L -	Ň L -	Ő L -	Ř L -	Ů L -	Ű L -	Ţ L -	ŕ L -	ă L -	Í L -	ć L -	č L -	ę L -	ě L -	ď L -
"5x	đ L -	ń L -	ň L -	ő L -	ř L -	ů L -	ű L -	ţ L -	˙ -	Ĉ L -	Ĝ L -	Ĥ L -	Ĵ L -	Ŝ L -	Ŭ L -	ĉ L -
"6x	ĝ L -	ĥ L -	ĵ L -	ŝ L -	ŭ L -	ɱ L -	ɳ L -	ɾ L -	ʃ L -	ʒ L -	ʈ L -	ʧ L -	ʉ L -	ʋ L -	ɖ L -	ɳ L -
"7x	ɾ L -	ʂ L -	ʐ L -	ɻ L -	ɭ L -	ʇ L -	ɲ L -	ʝ L -	ʎ L -	ɡ L -	ŋ L -	ɯ L -	ʁ L -	ħ L -	ʕ L -	

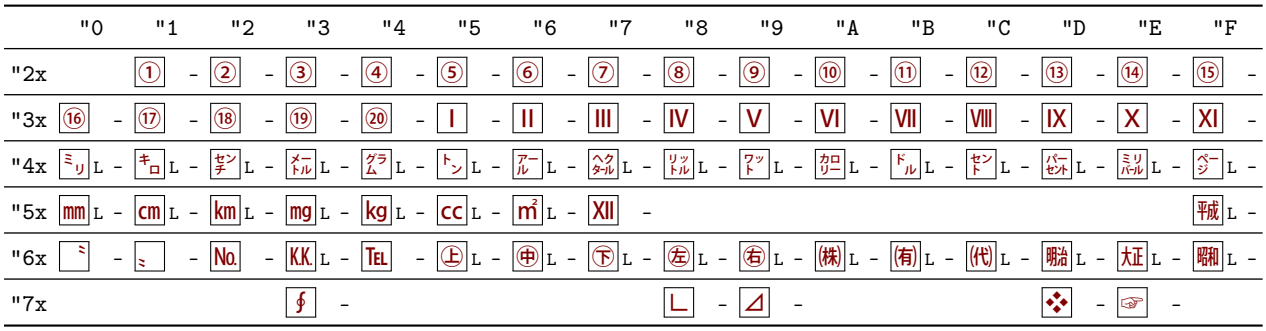
Row 11



Row 12



Row 13



B Package versions used in this document

This document was typeset using the following packages:

```
geometry.sty            2010/09/12 v5.6 Page Geometry
keyval.sty              1999/03/16 v1.13 key=value parser (DPC)
ifpdf.sty               2011/01/30 v2.3 Provides the ifpdf switch (HO)
ifvtex.sty              2010/03/01 v1.5 Detect VTeX and its facilities (HO)
ifxetex.sty             2010/09/12 v0.6 Provides ifxetex conditional
luatexja-adjust.sty     2013/05/14
luatexja.sty            2013/05/14 Japanese Typesetting with LuaTeX
luatexja-core.sty       2013/05/14 Core of LuaTeX-ja
luaotfload.sty          2013/05/23 v2.2d OpenType layout system
luatexbase.sty          2013/05/11 v0.6 Resource management for the LuaTeX macro programmer
ifluatex.sty            2010/03/01 v1.3 Provides the ifluatex switch (HO)
luatex.sty              2010/03/09 v0.4 LuaTeX basic definition package (HO)
infwarerr.sty           2010/04/08 v1.3 Providing info/warning/error messages (HO)
```

```
etex.sty                 1998/03/26 v2.0 eTeX basic definition package (PEB)
luatex-loader.sty        2010/03/09 v0.4 Lua module loader (HO)
luatexbase-compat.sty    2011/05/24 v0.4 Compatibility tools for LuaTeX
luatexbase-modutils.sty  2013/05/11 v0.6 Module utilities for LuaTeX
luatexbase-loader.sty    2013/05/11 v0.6 Lua module loader for LuaTeX
luatexbase-regs.sty      2011/05/24 v0.4 Registers allocation for LuaTeX
luatexbase-attr.sty      2013/05/11 v0.6 Attributes allocation for LuaTeX
luatexbase-cctb.sty      2013/05/11 v0.6 Catcodetable allocation for LuaTeX
luatexbase-mcb.sty       2013/05/11 v0.6 Callback management for LuaTeX
ltxcmds.sty              2011/11/09 v1.22 LaTeX kernel commands for general use (HO)
pdftexcmds.sty           2011/11/29 v0.20 Utility functions of pdfTeX for LuaTeX (HO)
xkeyval.sty              2012/10/14 v2.6b package option processing (HA)
ltj-base.sty             2013/05/14
ltj-latex.sty            2013/05/14 LaTeX support of LuaTeX-ja
lltjfont.sty             2013/05/14 Patch to NFSS2 for LuaTeX-ja
lltjdefs.sty             2013/06/12 Default font settings of LuaTeX-ja
lltjcore.sty             2013/05/14 Patch to LaTeX2e Kernel for LuaTeX-ja
luatexja-compat.sty      2013/05/14 Compatibility with pTeX
expl3.sty                2013/03/14 v4469 L3 Experimental code bundle wrapper
l3names.sty              2012/12/07 v4346 L3 Namespace for primitives
l3bootstrap.sty          2013/01/08 v4420 L3 Experimental bootstrap code
l3basics.sty             2013/01/10 v4428 L3 Basic definitions
l3expan.sty              2013/02/03 v4458 L3 Argument expansion
l3tl.sty                 2013/01/08 v4415 L3 Token lists
l3seq.sty                2013/01/12 v4434 L3 Sequences and stacks
l3int.sty                2013/01/13 v4444 L3 Integers
l3quark.sty              2012/11/04 v4268 L3 Quarks
l3prg.sty                2013/02/13 v4459 L3 Control structures
l3clist.sty              2013/01/08 v4414 L3 Comma separated lists
l3token.sty              2013/01/10 v4428 L3 Experimental token manipulation
l3prop.sty               2013/01/09 v4423 L3 Property lists
l3msg.sty                2013/01/08 v4412 L3 Messages
l3file.sty               2013/01/14 v4446 L3 File and I/O operations
l3skip.sty               2013/01/13 v4444 L3 Dimensions and skips
l3keys.sty               2013/02/24 v4461 L3 Experimental key-value interfaces
l3fp.sty                 2013/01/19 v4449 L3 Floating points
l3box.sty                2013/01/08 v4411 L3 Experimental boxes
l3coffins.sty            2012/09/09 v4212 L3 Coffin code layer
l3color.sty              2012/08/29 v4156 L3 Experimental color support
l3luatex.sty             2012/08/03 v4049 L3 Experimental LuaTeX-specific functions
l3candidates.sty         2013/03/14 v4468 L3 Experimental additions to l3kernel
amsmath.sty              2013/01/14 v2.14 AMS math features
amstext.sty              2000/06/29 v2.01
amsgen.sty               1999/11/30 v2.0
amsbsy.sty               1999/11/29 v1.2d
amsopn.sty               1999/12/14 v2.01 operator names
tikz.sty                 2010/10/13 v2.10 (rcs-revision 1.76)
pgf.sty                  2008/01/15 v2.10 (rcs-revision 1.12)
pgfrcs.sty               2010/10/25 v2.10 (rcs-revision 1.24)
everyshi.sty             2001/05/15 v3.00 EveryShipout Package (MS)
pgfcore.sty              2010/04/11 v2.10 (rcs-revision 1.7)
graphicx.sty             1999/02/16 v1.0f Enhanced LaTeX Graphics (DPC,SPQR)
graphics.sty             2009/02/05 v1.0o Standard LaTeX Graphics (DPC,SPQR)
trig.sty                 1999/03/16 v1.09 sin cos tan (DPC)
pgfsys.sty               2010/06/30 v2.10 (rcs-revision 1.37)
xcolor.sty               2007/01/21 v2.11 LaTeX color extensions (UK)
pgfcomp-version-0-65.sty 2007/07/03 v2.10 (rcs-revision 1.7)
```

```
pgfcomp-version-1-18.sty 2007/07/23 v2.10 (rcs-revision 1.1)
pgffor.sty               2010/03/23 v2.10 (rcs-revision 1.18)
pgfkeys.sty
pict2e.sty               2011/04/05 v0.2y Improved picture commands (HjG,RN,JT)
multienum.sty
float.sty                2001/11/08 v1.3d Float enhancements (AL)
booktabs.sty             2005/04/14 v1.61803 publication quality tables
multicol.sty             2011/06/27 v1.7a multicolumn formatting (FMi)
listings.sty             2007/02/22 1.4 (Carsten Heinz)
lstmisc.sty              2007/02/22 1.4 (Carsten Heinz)
showexpl.sty             2013/03/21 v0.3k Typesetting example code (RN)
calc.sty                 2007/08/22 v4.3 Infix arithmetic (KKT,FJ)
ifthen.sty               2001/05/26 v1.1c Standard LaTeX ifthen package (DPC)
varwidth.sty             2009/03/30 ver 0.92;  Variable-width minipages
hyperref.sty             2012/11/06 v6.83m Hypertext links for LaTeX
hobsub-hyperref.sty      2012/05/28 v1.13 Bundle oberdiek, subset hyperref (HO)
hobsub-generic.sty       2012/05/28 v1.13 Bundle oberdiek, subset generic (HO)
hobsub.sty               2012/05/28 v1.13 Construct package bundles (HO)
intcalc.sty              2007/09/27 v1.1 Expandable calculations with integers (HO)
etexcmds.sty             2011/02/16 v1.5 Avoid name clashes with e-TeX commands (HO)
kvsetkeys.sty            2012/04/25 v1.16 Key value parser (HO)
kvdefinekeys.sty         2011/04/07 v1.3 Define keys (HO)
pdfescape.sty            2011/11/25 v1.13 Implements pdfTeX's escape features (HO)
bigintcalc.sty           2012/04/08 v1.3 Expandable calculations on big integers (HO)
bitset.sty               2011/01/30 v1.1 Handle bit-vector datatype (HO)
uniquecounter.sty        2011/01/30 v1.2 Provide unlimited unique counter (HO)
letltxmacro.sty          2010/09/02 v1.4 Let assignment for LaTeX macros (HO)
hopatch.sty              2012/05/28 v1.2 Wrapper for package hooks (HO)
xcolor-patch.sty         2011/01/30 xcolor patch
atveryend.sty            2011/06/30 v1.8 Hooks at the very end of document (HO)
atbegshi.sty             2011/10/05 v1.16 At begin shipout hook (HO)
refcount.sty             2011/10/16 v3.4 Data extraction from label references (HO)
hycolor.sty              2011/01/30 v1.7 Color options for hyperref/bookmark (HO)
auxhook.sty              2011/03/04 v1.3 Hooks for auxiliary files (HO)
kvoptions.sty            2011/06/30 v3.11 Key value format for package options (HO)
url.sty                  2006/04/12  ver 3.3  Verb mode for urls, etc.
rerunfilecheck.sty       2011/04/15 v1.7 Rerun checks for auxiliary files (HO)
amsthm.sty               2004/08/06 v2.20
luatexja-otf.sty         2013/05/14
luatexja-ajmacros.sty    2013/05/14
luatexja-preset.sty      2013/05/25 Japanese font presets
luatexja-fontspec.sty    2013/05/14 fontspec support of LuaTeX-ja
fontspec.sty             2013/05/20 v2.3c Font selection for XeLaTeX and LuaLaTeX
xparse.sty               2013/03/12 v4467 L3 Experimental document command parser
fontspec-patches.sty     2013/05/20 v2.3c Font selection for XeLaTeX and LuaLaTeX
fixltx2e.sty             2006/09/13 v1.1m fixes to LaTeX
fontspec-luatex.sty      2013/05/20 v2.3c Font selection for XeLaTeX and LuaLaTeX
fontenc.sty
xunicode.sty             2011/09/09 v0.981 provides access to latin accents and many other characters in Unicode
                         lower plane
unicode-math.sty         2013/05/04 v0.7e Unicode maths in XeLaTeX and LuaLaTeX
l3keys2e.sty             2013/03/12 v4467 LaTeX2e option processing using LaTeX3 keys
catchfile.sty            2011/03/01 v1.6 Catch the contents of a file (HO)
fix-cm.sty               2006/09/13 v1.1m fixes to LaTeX
filehook.sty             2011/10/12 v0.5d Hooks for input files
unicode-math-luatex.sty
lualatex-math.sty        2013/01/13 v1.2 Patches for mathematics typesetting with LuaLaTeX
```

```
etoolbox.sty             2011/01/03 v2.1 e-TeX tools for LaTeX
metalogo.sty             2010/05/29 v0.12 Extended TeX logo macros
lltjp-fontspec.sty       2013/05/14 Patch to fontspec for LuaTeX-ja
lltjp-xunicode.sty       2013/05/14 Patch to xunicode for LuaTeX-ja
lltjp-unicode-math.sty   2013/05/14 Patch to unicode-math for LuaTeX-ja
lltjp-listings.sty       2013/05/14 Patch to listings for LuaTeX-ja
epstopdf-base.sty        2010/02/09 v2.5 Base part for package epstopdf
grfext.sty               2010/08/19 v1.1 Manage graphics extensions (HO)
nameref.sty              2012/10/27 v2.43 Cross-referencing by name of section
gettitlestring.sty       2010/12/03 v1.4 Cleanup title references (HO)
```